Semantic-based Model Matching with EMFCompare

Lorenzo Addazi
Mälardalen University, IDT,
Västerås, Sweden
lai15004@student.mdh.se

Antonio Cicchetti
Mälardalen University, IDT,
Västerås, Sweden
antonio.cicchetti@mdh.se

Juri Di Rocco
University of L'Aquila
I-67100 L'Aquila, Italy
juri.dirocco@univaq.it

Davide Di Ruscio
University of L'Aquila
I-67100 L'Aquila, Italy
davide.diruscio@univaq.it

Ludovico Iovino
Gran Sasso Science Institute
I-67100 L'Aquila, Italy
ludovico.iovino@gssi.infn.it

Alfonso Pierantonio
University of L'Aquila
I-67100 L'Aquila, Italy
Mälardalen University, IDT,
Västerås, Sweden
alfonso.pierantonio@univaq.it,
alfonso.pierantonio@mdh.it

ABSTRACT

In MDE resolving pragmatic issues related to the management of models is key to success. Model comparison is one of the most challenging operations playing a central role in a wide range of modelling activities including model versioning, evolution and even collaborative and distributed specification of models. Over the last decade, several syntactic methods have been proposed to compare models even though they struggle in achieving higher levels of accuracy especially when the semantics of the application domain has to be considered. Existing methods improve comparison precision at the price of high performance costs.

This paper discusses a lightweight semantic comparison method, which relies on a new matching algorithm that considers ontological information encoded in the WordNet lexical database further than ordinary syntactical and structural correlations. The approach has been implemented as extension of EMFCompare and evaluated to measure its precision and performances when compared to existing approaches.

Keywords

model differencing; syntactic matching; semantic matching; ontological matching; EMFCompare

1. INTRODUCTION

Model-Driven Engineering (MDE) promotes the migration from a code-centric to a model-based approach to cope with the increasing development complexity of modern software systems. Models abstract real-world phenomena focusing on a specific aspect, e.g. its dynamic behaviour, its static structure, and assume the role of first-class artefacts throughout the software life cycle [1].

Addressing pragmatic issues related to the management and evolution of models has become a major concern in Model Driven Engineering [2] (MDE). Critical facilities such as model mergers and model difference tools related to model-level observability are key in distributed development of modeling artifacts. As noticed already in 2003 by Bran Selic model difference tools *"must work at a semantically meaningful level."* [3].

Over the years, relevant advances have been provided in the area of model differencing in terms of methods, such as similarity-based approaches [4, 5], and tools with the introduction of the EMFCompare [6] in the EMF ecosystem. However, the problem of determining model differences represents an intrinsically complex task, especially when dealing with two-way state-based comparisons [7], i.e. when the differencing is based on the sole information stored in the two models being compared.

Model differencing, indeed, relies on *Model Matching*, i.e. the calculation of correspondences among model elements, which can be reduced to the NP-Hard *Graph Isomorphism Problem*, that is the problem of finding correspondences between graphs [8, 5]. The available approaches to model matching are all different ways to deal with this intrinsic hardness, which is typically alleviated through either domain-specific or generic but approximate solution [9].

In this paper, we make a first step towards the introduction of semantic reasoning within the matching process in EMFCompare. In particular, we present a custom matching engine extending the default one of EMFCompare. In addition to the syntactical and structural correlations, the proposed extension compares model elements with respect to their semantic meaning using the WordNet lexical database [10]. Furthermore, the effectiveness and efficiency of the proposed approach is evaluated on a matching scenario based on an existing benchmark.

Structure of the paper: The paper is organized as follows. Section 2 presents an example motivating the work. Section 3 provides an overview of the model differencing problem. Section 4 outlines the main phases of the comparison process of EMFCompare and highlights its limitations. Section 5 outlines the WordNet lexical database, and the various semantic similarity measures based on its structural organization of concepts. Section 6 presents our main contribution, that is, a semantic extension of the matching process in EMFCompare. Section 7 evaluates the extension on a matching scenario based on an existing benchmark. Section 8 concludes the paper providing a brief summary and a discussion of possible future directions.

2. MOTIVATING SCENARIO

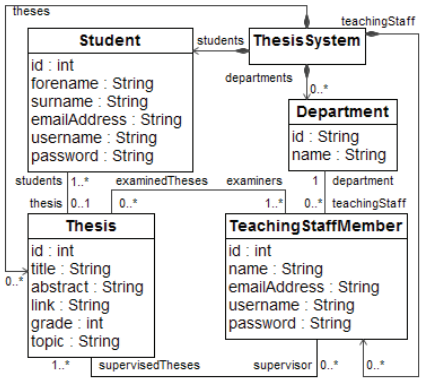
As mentioned before, the problem of calculating differences between two model versions is intrinsically difficult. Figure 1 depicts a practical example involving two different versions of the *ThesisManagementSystem* metamodel, that is a small-scale reproduction of university theses management portal. According to the initial version of the metamodel shown in Fig.1.a, a `ThesisSystem` contains information about `Departments`, `Students`, teaching staff (`TeachingStaffMember`), and `Thesis`. In order to satisfy unforeseen requirements or to refine existing constructs, metamodels can be modified as in the case of the new version of the *ThesisManagementSystem* metamodel shown in Fig.1.b, which has

been obtained by applying the two metamodel changes described below.

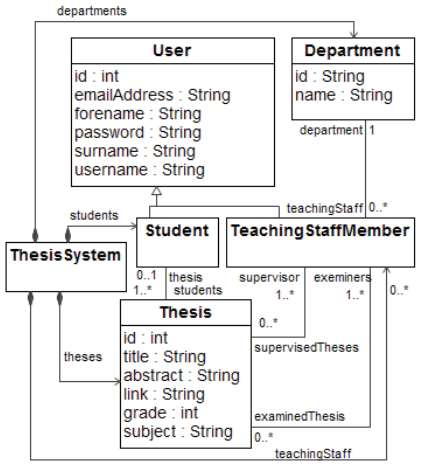
Extract user super class: in the first version of the metamodel, `Student` and `TeachingStaffMember` classes are completely separated despite sharing most of their attributes. Moreover, the metaclass `Student` has `forename` and `surname` attributes while `TeachingStaffMember` has only a `name` attribute grouping both name and surname. In light of this, the evolved metamodel version extracts the common information among `Student` and `TeachingStaffMember` into an additional superclass `User`, while `TeachingStaffMember`'s `name` is partitioned in `User`'s `forename` and `surname` attributes.

Thesis Attribute Renaming: in the first version of the metamodel, the main theme of a thesis is expressed through the `topic` attribute in the `Thesis` class. In the newer version, this attribute is renamed to `subject`. Intuitively, these attributes express the same concept, i.e. the principal issue discussed in a given thesis.

Table 1 reports the desired correspondences between the two metamodels, as determined by manually comparing the two versions. In particular, the entries named `S`, `T`, `D`, `Th`, `U` and `TS` correspond to `Student`, `TeachingStaffMember`, `Department`, `Thesis`, `User`, and `ThesisSystem` entities in the metamodels. For instance, there is a match between `Student` class in the initial and evolved metamodels; between the attributes `id`, `forename`, etc. in the initial metamodel, and the corresponding ones inherited by the class `User`, i.e. `U.id`, `U.forename`, etc. in the evolved version; between the relationships outgoing from `Student` and targeting the `Thesis` classes. The total number of manually identified correspondences is 37.



a) Initial metamodel version



b) Evolved metamodel version

Figure 1: A (meta-)model evolution scenario

Table 1: Manually identified matches

Element version 1	Element version 2
`S`	`S`
`S.id`	`U.id`
`S.forename`	`U.forename`
`S.surname`	`U.surname`
`S.emailAddress`	`U.emailAddress`
`S.username`	`U.username`
`S.password`	`U.password`
`S.thesis`	`S.thesis`
`T`	`T`
`T.id`	`U.id`
`T.name`	`U.forename`
`T.name`	`U.surname`
`T.emailAddress`	`U.emailAddress`
`T.username`	`U.username`
`T.password`	`U.password`
`T.examinedTheses`	`T.examinedTheses`
`T.supervisedTheses`	`T.supervisedTheses`
`T.department`	`T.department`
`D[4 matches]`*	`D[4 matches]`*
`Th[9 mathches]`•	`Th[9 matches]`•
`Th.topic`	`Th.subject`
`TS[5 matches]`*	`TS[5 matches]`*

* contained structural features are fully matched
• remaining structural features are fully matched

3. MODEL DIFFERENCING OVERVIEW

Existing model differencing approaches can be considered as logically composed of three main phases: *i)* compared models are imported in a differencing friendly format (typically graph-based structures); *ii)* a matching algorithm navigates the models to detect and establish correspondences between entities in the two models being compared; *iii)* a dedicated algorithm computes the differences of the matched elements in terms of (at least) additions, deletions, and changes of model entities as based on the matches established in the previous phase. In this respect, the matching phase is critical in any differencing approach, since any erroneous match, both false-positive and false-negative, results in a wrong output. More in general, requirements for model matching approaches include accuracy, a high level of abstraction at which the comparison is performed, independence from particular tools, domains, and languages, efficiency, and minimal effort. In [9] authors classify model matching approaches as follows:

– *Static Identity-Based Matching*: in this category, it is assumed that each model element has a persistent unique identifier that is assigned to it upon creation. Therefore, a basic approach for matching models is to identify matching model elements based on their corresponding identities (as in [11, 12, 13]);

– *Signature-Based Matching*: in this category, the identity of each model element is not static. Instead its identity, typically referred to as signature, is dynamically calculated by combining the values of its features. The signature computa-

tion is performed by means of a user-defined function specified using a model querying language (as in [14]);

– *Similarity-Based Matching*: the approaches in this category treat models as typed attribute graphs and attempts to identify matching elements based on the aggregated similarity of their features. It is worth noting that not all features of model elements are equally relevant for establishing a match (e.g. classes with matching names are more likely to be matched with classes specialising the same parent superclass). Therefore, similarity-based algorithms typically need to be provided with a configuration that specifies the relative weight of each feature and thus of detected correspondences (as in [5, 15, 13]);

– *Custom Language-Specific Matching Algorithms*: this category involves matching algorithms tailored to a particular modelling language. Notably, UMLDiff [4] and the work in [16] specifically target UML models and statecharts, respectively. In these cases, the narrow set of available entities, their semantics, and their valid evolution alternatives, allows to specialise the matching algorithm computations.

None of the approaches previously listed can be considered as the best solution as shown in [9]. Instead, the choice of a model matching solution should be evaluated as trying to optimise the trade-off between the constraints imposed by the context and the particular task at stake. This work is based on EMFCompare due to the flexibility and customisability of its matching engine, as detailed in the remainder of this section.

4. THE EMFCOMPARE TOOL: OVERVIEW AND LIMITATIONS

EMFCompare [13] is an Eclipse project which was initiated in 2006 at Eclipse Summit Europe, where the need for a model comparison engine emerged. It provides generic and customisable support for model comparison and merge of EMF models, such as Ecore and UML. The most relevant characteristic with respect to other approaches, at the time, consists in its high degree of extensibility: in fact, the whole comparison process is indeed designed to be completely customisable. In particular, the different activities composing the comparison process are explicitly disjoint and managed using different software entities, i.e. *engines*. Furthermore, according to its extensible nature, EMFCompare provides full support for the extension of its default metamodel-independent behaviour to tailor custom or metamodel-specific solutions.

The main advantage resulting from the explicit partition of the comparison process in its various sub-activities is that developers can focus on the specific part of the process they are going to extend/customise, while leaving the management of the remaining steps to the framework. For instance, the default matching approach adopted in EMFCompare falls in the *similarity-based* techniques. However, it also provides the possibility to adopt a *static identity-based* or a *signature-based* matching approach, possibly defining custom generator functions, if needed. In this regard, it is worth mentioning that this paper focuses on metamodel-independent differencing, which is incompatible with *static identity-based* and *signature-based* approaches [17].

In the following, we first provide a brief description of the overall comparison process in EMFCompare. Then, we focus our attention on the issues and limitations affecting the matching phase; such a phase will constitute the foothold on which our main contribution is built.

4.1 The Comparison Process

The EMFCompare comparison process can be roughly divided into six phases, whose main characteristics are singularly described below. The matching phase is illustrated in deeper details due to its relevance for this contribution, while the interested reader can refer to [13] for further details about the other phases.

Phase 1: Model resolving - This phase builds a logical representation of the overall comparison context, i.e. a model representing the artefacts to be compared enriched with the information required for the subsequent activities.

Phase 2: Model matching - Once resolved the logical models, the matching phase creates a set of *two-by-two* mappings while iterating over the model elements, e.g. `Student` in the model in Fig.1.a corresponds to *Student* in the model in Fig.1.b. By going into more details, for each element in the first model it is necessary to browse the elements in the second in order to find the most similar one. The default match engine firstly selects a specific element pair, and subsequently a similarity evaluation is computed by the combination of four different metrics in an overall score ranging from 0, i.e. completely different, to 1, i.e. identical elements. These metrics analyse the name of an element, its content, its type, and its relations with other elements.

Despite considering different characteristics of a given element, all the metrics adopt the same comparison approach. Indeed, for each metric, the final result is given by the execution of a string distance algorithm, e.g. the *Levenshtein Distance Algorithm* [18], on the string representation of the elements. In order to better explain this fundamental phase, listing 1 reports a simplified matching-algorithm pseudo-code. After the result initialisation (line 1), the algorithm iterates over all possible element pairs to compute their similarity (lines 2–4). It is worth noting that for optimisation purposes the models are not compared as a whole, whereas the possible elements are selected within a proper search window (line 3).

Once all the possible candidate matches are identified, the *createMatch* method takes as input the similarity values, and by using threshold policies produces the *Comparison* object containing all detected matches. The elements included in the current search window which are not matched yet are forwarded to future searches to possibly produce new matches.

Listing 1: EMFCompare default match engine implementation

```
1 result = Double[Model1.getElements().size()][Model1.
      getElements().size()]
2 foreach (elM1 : Model1.getElements())
3   foreach (elM2 : elM1.getWindowElements())
4     result[elM1][elM2] = calculateSimilarity(elM1, elM2)
5 return createMatches(result)
```

Phase 3: Model differencing - In this phase, the output of the matching is analysed to classify the various changes happened from a version to another. For instance, an element only present in the old version is classified as a deletion, an element only present in the new version as an addition, while a match can either be an untouched element, or an element subject to updates.

Phase 4: Detection of difference equivalences - During the equivalences step, the produced differences are analysed for filtering out redundant correspondences.

Phase 5: Detection of difference requirements - In this phase the produced differences are analysed once more to detect possible dependencies among them. Notably, the addition of the specialisation relationship between `Student` and `User` in the model in Fig.1.b could not exist without the creation of `User`, which does not exist in the old version of the model (see Fig.1.a).

Table 2: Matches identified by EMFCompare

Element version 1	Element version 2
S	S
S.surname	U.surname
S.thesis	S.thesis
T	T
T.examinedTheses	T.examinedTheses
T.supervisedTheses	T.supervisedTheses
T.department	T.department
D[4 matches]*	D4 matches]*
Th [9 matches]•	Th [9 mathes]•
TS[5 matches]*	TS[5 matches]*

* contained structural features are fully matched
• topic and subject attributes are not paired but other contained structural features are fully matched.

Phase 6: Detection of difference conflicts - This phase allows to link the comparison process with a conflict detection mechanism, for instance when dealing with models managed through a Version Control System (VCS).

4.2 Limitations and Issues

According to performed experiments EMFCompare might have difficulties when dealing with comparison scenarios presenting complex correspondences. In particular, we define *complex* those differences which cannot be detected considering syntactical features only, or those differences characterised by n-to-m matches. Examples of representative complex correspondences are renaming an element using a synonym term, extracting features in common among multiple entities into a more generic one, or vice-versa, decomposing or distributing an entity feature among more specific entities.

By considering the default implementation of the EMFCompare matching process, we have classified the emerging issues into two categories, namely *contextual issues* and *linguistic issues*. The former result from the limited consideration of the features characterising the elements surrounding the compared ones, e.g. the classes containing a given pair of attributes. The latter, i.e. *linguistic issues*, result from the lack of a semantical evaluation of the features characterising the compared elements. Renaming a given class using a syntactically different name, for example, could lead to a false-negative, i.e. undetected correspondence. Analogously, a false positive, i.e. unexpected correspondence, could result when renaming a given class using a semantically different term, which however presents a strong syntactical similarity with another existing one.

Keeping in mind *contextual* and *linguistic* issues, Table 2 show the calculated matches with respect to the motivating example introduced in Sec.2. The EMFCompare default implementation identifies 25 matches, in particular 12 false negatives and 0 false positives. In particular, EMFCompare is not able to detect the correspondence between the attributes in the old `Student` metaclass and the new ones inherited from the extracted superclass `User`. Moreover, the comparison does not match the old `topic` attribute with the new `subject` attribute contained in `Thesis` metaclass.

5. WORDNET-BASED SIMILARITY

In this section, we provide an overview of the WordNet lexical database (see Sect. 5.1) and of related semantic similarity measures (see Sect. 5.2). Such techniques and tools underpin the improvement of EMFCompare proposed in Sect. 6 in order to mitigate the issues discussed in the previous section.

5.1 WordNet in a nutshell

WordNet [10] is a lexical database for the English language. It was created and is being maintained at the Cognitive Science Laboratory of Princeton University under the direction of psychology professor George A. Miller. In this database, English words are grouped into sets of synonyms called *synsets*, which also include a generic definition joining the contained words together and information about the semantic relationships connecting them to other synsets. The specific meaning of one word under a specific *Part-Of-Speech* (POS) is called a *sense*. Each synset has a *gloss* that defines the concept it represents. For example, the words *night*, *nighttime*, and *dark* constitute a single synset that has the following gloss: *"the time after sunset and before sunrise while it is dark outside"*. The purpose is twofold: to produce a combination of dictionary and thesaurus that is more intuitively usable, and to support automatic text analysis and artificial intelligence applications [19]. Synsets are connected to one another through explicit semantic relations. Some of these relations (hypernym, hyponym for nouns, and hypernym and troponym for verbs) constitute *is-a-kind-of* (holonymy) and *is-a-part-of* (meronymy for nouns) hierarchies. For example, tree is a kind of plant, tree is a hyponym of plant, and plant is a hypernym of tree. Analogously, trunk is a part of a tree, and we have trunk as a meronym of tree. While semantic relations apply to all members of a synset, because they share a meaning but are all mutually synonyms, words can also be connected to other words through lexical relations, including antonyms (i.e., opposites of each other) which are derivationally related, as well. WordNet provides also the polysemy count of a word, i.e. the number of synsets that contain the word. If a word participates in several synsets (i.e., has several senses) then typically some senses are much more common than others.

5.2 Semantic Similarity Measures

Semantic similarity measures might be used for performing tasks such as term disambiguation [20], as well as text segmentation, and for checking ontologies for consistency or coherence. All the currently available measures can be grouped into four classes [19]: i) *path-length* based, ii) *information-content* based, iii) *feature* based, and iv) *hybrid* measures. In the following, a brief explanation for each of these measure classes is provided with further emphasis on information-content based measures, which have been adopted to develop the proposed EMFCompare extension.

Path-length based measures. The main idea of path length based measures is that the similarity between two concepts is a function of the length of the path linking the concepts and the position of the concepts in the WordNet taxonomy. Although most path length based measures are simple to use, local density of pairs (i.e., frequency of the involved terms) fails to be reflected [21].

Feature based measures. Unlike the other measures, feature based measures are independent from the taxonomy and the subsumptions of the concepts. In particular, they attempt to exploit the properties of the ontology in order to obtain similarity values. Indeed, this kind of measures is based on the assumption that each concept is described by a set of words indicating its properties or features, such as glosses in WordNet. The more common characteristic two concepts have and the less non-common characteristics they have, the more similar the concepts are. However, it is worth noting that this kind of measures introduces a noteworthy computational delay into the overall process. Furthermore, feature based measures require a complete and correct feature set to work properly, which is not always an easy task to perform [21].

Information-content based measures. In information-content (IC) based measures, it is assumed that each concept includes a certain amount of information in WordNet. Similarity measures are based on such information content of each concept within the taxonomy. The more common information two concepts share, the more similar the concepts are. Lin's Measure [22] uses both the amount of information needed to state the commonality between the two concepts and the information needed to fully describe these terms. The similarity measure formula is defined as follows:

$$Sim_{Lin}(c_1, c_2) = \frac{2 * IC(lso(c_1, c_2))}{IC(c_1) + IC(c_2)} \quad (1)$$

where $IC(lso(c_1, c_2))$ is the information needed to state the commonality between c_1 and c_2, whereas $IC(c_i)$ is the information needed to represent the concept c_i.

In general, IC based measures attempt to exploit the information related to a given pair of concepts in order to evaluate their similarity. Therefore, how to obtain IC represents a crucial issue, which will directly affect the measure application performance. In [23], the IC value of a concept is a function of its number of hyponyms. The more hyponyms a concept has, the more abstract it is. That is to say, concepts with many hyponyms convey less information than concepts that are leaves in the taxonomy. A common issue encountered in adopting IC based measures consists in less precise evaluations as regards the structure information of concepts, which is fairly captured in path length based measures on the contrary. However, IC based measures provide a relevant improvement in terms of computational complexity, thus making it the convenient in our case.

Hybrid measures. The hybrid measures combine the above mentioned ones. In practice, many measures not only are able to combine the ideas presented above, but also the relations, such as is-a, part-of, and so on. For instance, the Rodriguez's measure [24] includes three parts: (i) synonyms set, (ii) neighbourhoods, and (iii) features. The similarity value of each part is assigned to a weight, and then summed together. Generally, both IC and path length based measures are integrated as parameters into hybrid functions, as it is has been proposed in [25].

6. EXTENSION OF EMFCOMPARE WITH SEMANTIC MODEL MATCHING

In this section, we propose an extension of EMFCompare aiming at addressing both the *linguistic* and *contextual* issues presented in Section 4.2. Furthermore, our solution introduces support for the definition of custom matching approaches using ontological descriptions in EMFCompare, instead of custom-tailored match engine implementations.

Thanks to the intrinsic extensibility of EMFCompare, our method has been developed focusing on the matching phase only, hence leaving untouched the previous and subsequent steps[1]. In particular, the matching process has been modified redefining the selection process of model elements, i.e. which elements to compare among each other, and the evaluation approach itself, i.e. the computation of the similarity value corresponding to a specific model elements pair. The proposed solution limits its semantic reasoning to the comparison among model element names, and relies on the WordNet lexical dictionary as ontological source. However, the knowledge is not directly retrieved from the WordNet database, but rather from an automatically generated graph. In this way, our method does not directly depend on the WordNet database structure itself, hence enabling the adoption of other ontologies.

In order to define our method, we have extended the concept of *Match* in the underlying EMFCompare comparison model by introducing the *Semantic Match* concept (see the `Semantic Match Engine` element in Fig. 2). The differences with respect to the ordinary *Match* consist in: (i) *semantic distance value*, i.e. each match carries information about the semantic distance among the encapsulated model elements, (ii) *container matches list*, i.e. each match carries references to the containers of the encapsulated model elements, and (iii) *content matches list*, i.e. each match carries references to the content of the encapsulated model elements.

The overall matching process, as illustrated in Listing 2, can be decomposed in three separated phases: (i) *exploration*, (ii) *evaluation* and (iii) *filtering*. In compliance with the default implementation in EMFCompare, the process starts receiving a logical representation of the compared models and terminates producing a set containing the expected matches.

```
1 function createMatches(Comparison comparison, List
      leftEObjects, List rightEObjects){
2  SemanticMatch root = createSemanticMatch(null, null);
3  exploreMatches(root, leftEObjects, rightEObjects);
4  evaluateMatches(root);
5  filterMatches(root, comparison);
6 }
```

Listing 2: Pseudo-code of the proposed semantic model match approach

The three steps underpinning the proposed semantic match are singularly described in the following.

Exploration: during the exploration phase, our method builds a labelled graph representation of the compared models. In such a representation, each node represents a *semantic match*, while each incoming or outgoing labelled edge represents a connection with its parents or children elements, respectively. The exploration process is listed in Listing 3. Given a starting pair of model elements, i.e. the `root` parameter, the exploration phase first iterates over their contained elements, (lines 2-3). For each pair of elements of the same type, our method creates a new `Semantic Match` node and connects it to the initial pair (lines 5-7). Finally, the same exploration process is recursively repeated on the created node (line 8).

```
1 function exploreMatches(SemanticMatch root, List
      leftEObjects, List rightEObject){
2  foreach(leftEObject : leftEObjects)
3   foreach(rightEObject : rightEObjects){
4    if(leftEObject.getClass().equals(rightEObject.getClass
        ())){
5     SemanticMatch current = createSemanticMatch(
         leftEObject, rightEObject);
6     current.addParent(root);
7     root.addChild(current);
8     exploreMatches(current, leftEObject.getContent(),
         rightEObject.getContent());
9    }
10  }
11 }
```

Listing 3: Pseudo-code of the Exploration phase

Evaluation: once the graph representation of the compared models is created, each `Semantic Match` node is integrated with the semantic distance value between its encapsulated elements, as illustrated in Listing 4. Once completed the evaluation of a given node (line 2), the same process is recursively repeated on its children elements (lines 4-6).

[1]The prototypical implementation of the EMFCompare semantic model matching extension is available at: https://github.com/MDEGroup/EMFCompare-Semantic-Extension .

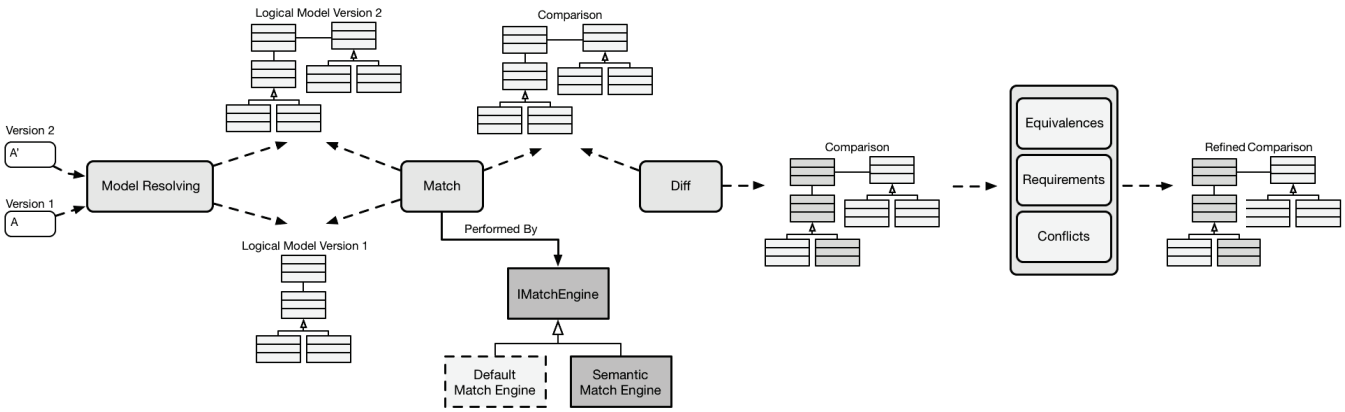


Figure 2: The proposed `Semantic Match Engine` in EMFCompare

```
1 function evaluateMatches(SemanticMatch root){
2  Double semanticDistance = evaluateSemanticDistance(root.
       getLeft(), root.getRight());
3  root.setSemanticDistance(semanticDistance);
4  foreach(child : root.getChildren()){
5   evaluateMatches(child);
6  }
7 }
```

Listing 4: Pseudo-code of the Evaluation phase

The semantic distance evaluation process is described in Listing 5. Given two model elements, the process begins with three subsequent preprocessing operations aimed to prepare the input for the subsequent evaluation (lines 2-3). Each string is tokenized by using as delimeters any non-alphabetical character, blank spaces and uppercasing - lowcasing changes in the word (line 12). Once, the obtained sequences are analyzed using the *Stanford Log-Linear Part-Of-Speech Tagger* [26], which assigns parts of speech, such as noun, verb or adjective, to each elements (line 13). Finally, in the token stemming step, for each tagged token, our method retrieves the corresponding root word using WordNet (line 14). At the end of the preprocessing phase, the initial strings are transformed to POS-tagged sequences, which represent the input for the actual comparison phase.

In order to evaluate the semantic similarity between two token sequences, our method first compares each element from the first with the elements from the latter (lines 5-6). For each element pair, the comparison consists in retrieving all the possible synsets which are related to the given words, evaluate their semantic distance using the inverted Lin's Algorithm (see Sect. 5.2), and returns back the minimum obtained value, i.e. how much the given words are similar in the best case (lines 7). In particular, we chose to adopt the corpora-independent method proposed in [23], which uses WordNet itself as a statistical resource to calculate IC values. Once that each pair has been evaluated, the final result is given by the *Min Match Average* of the obtained values, i.e. the sum of the minimum distance for each pair of tokens, divided by the maximum token list length among the involved ones (line 8). The final value ranges from 0, i.e. identical elements, to 1, i.e. not matching.

```
1 function Double evaluateSemanticDistance(EObject left,
      EObject right){
2  leftTokens = inputProcessing(left);
3  rightTokens = inputProcessing(right);
4  Double[][] result = new Double[leftTokens.size()][
       rightTokens.size()];
5  foreach(leftToken : leftTokens)
6   foreach(rightToken : rightTokens)
7    result[leftToken][rightToken] = invertedLin(leftToken,
         rightToken);
8  return minMatchAverage(result);
9 }
10
11 function List inputProcessing(EObject element){
12  List result = tokenizeString(element.getName());
13  result = tagTokenList(result);
14  return stemTokenList(result);
15 }
```

Listing 5: Pseudo-code of the semantic distance evaluation

Filtering: starting from the semantic distance values obtained during the previous steps, in this phase each `Semantic Match` node is analysed in order to decide whether or not to put it into the result set, as illustrated in Listing 6. Given an initial node, its analysis value results from the weighted arithmetic mean of the semantic distance value between the encapsulated model elements, the average semantic distance values between their children, and the minimum semantic distance value between their parents (lines 2-5). In order to be accepted, a `Semantic Match` node must have a lower analysis value with respect to a pre-defined threshold T_α. According to the requirements imposed by EMFCompare, the final matches are inserted into the *comparison* set (line 7). Furthermore, for each rejected node, we create two substituting matches having the left and the right element only, respectively (lines 9-10). The same process is then recursively repeated for each `Semantic Match` child (lines 12-14).

```
1 function filterMatches(SemanticMatch root, Comparison
      comparison){
2  Double rootDistance = root.getSemanticDistance() * root.
       getSemanticDistanceWeight();
3  Double childrenDistance = averageSemanticDistance(root.
       getChildren());
4  Double parentDistance = minSemanticDistance(root.
       getParents());
5  Double overallDistance = weightedArithmeticMean(
       semanticDistance, childrenDistance, parentDistance)
       ;
6  if(overallDistance < T_α){
7   comparison.add(root);
8  } else {
9   comparison.add(new SemanticMatch(root.getLeft(), null))
        ;
10  comparison.add(new SemanticMatch(null, root.getRight())
        );
11 }
12 foreach(child : root.getChildren()){
13  filterMatches(child, comparison);
14 }
15 }
```

```
16
17function Double weightedArithmeticMean(Double root,
      Double children, Double parent){
18  return W_R*root + W_C*children + W_P*parent;
19}
```

Listing 6: Pseudo-code of the Filtering phase pseudo-code

By considering the motivating example presented in Section 2, the proposed semantic approach identifies correctly the 37 manual correspondences and no *false positives* are found. Therefore, the calculated matches are identical to the manual correspondences and the first matches are exactly the same shown in Table 1.

7. VALIDATION

In order to validate our approach, we performed an experiment using a matching case benchmark inspired by [27] beyond successful tests on the motivating example presented in Section 2. This choice was due to the impossibility to reproduce our example on [27], and also to the opportunity of exploiting a larger experimental set. Once completed the experiment, we have compared our results to the principal state-of-the-art model matching tools. In detail, we have considered EMFCompare and *Atlas Model Weaver* (AMW), which represent the state-of-the-art matching tools in EMF. Furthermore, in order to extend our evaluation to matching tools in general, we have also included four existing approaches from the field of ontology and schema matching [28, 29, 30, 31], and the search-based approach proposed in [27]. Furthermore, we have compared our approach against *GAMMA* and [27] with respect to the difference computation performances.

In this section, we first describe the corpus of data used in our experiment, as well as the measures used for the evaluation. Then, we discuss the obtained results and compare them to the results produced using the other approaches. We refer the interested reader to [27] for further details about the results obtained by the other tools.

7.1 The Model Exchange Benchmark

The benchmark considered for our experiment consists of five structural modelling languages, namely UML 2.0, UML 1.4.2, Ecore, WebML and EER, as shown in Table 3. These metamodels present different characteristics with respect to both the size and used terminology. Indeed, as far as the metamodel size is concerned, the considered metamodels range from small-sized, e.g. EER, to large-sized, e.g. UML 2.0. Furthermore, while Ecore and UML metamodels use an object-oriented (OO) terminology, WebML and EER use database (DB) terminology.

All the possible pairs of the considered metamodels are considered as input to calculate the matches by means of GAMMA, EMFCompare, and the proposed approach named Semantic EMFCompare hereafter. In order to evaluate the quality of the produced match results, manual correspondences are reused from the previous studies using the INRIA alignment format provided in [27].

7.2 Measures

In order to evaluate the results produced by our approach, we consider them with respect to three different metrics from the information retrieval field: *Precision*, *Recall* and *F-Measure*. In this context, the *Precision* measure denotes the percentage of correctly matched elements with respect to all the proposed matchings as shown in equation 2.

$$Precision = \frac{|\{Retrieved\,Matches\} \cap \{Relevant\,Matches\}|}{|\{Retrieved\,Matches\}|} \quad (2)$$

Metamodel	Size	Terminology
UML 2.0 CD	158	OO
UML 1.4.2 CD	143	OO
Ecore	83	OO
WebML	53	DB
EER	23	DB

Table 3: Metamodels of the Model Exchange benchmark

The *Recall* measure, instead, indicates the percentage of correctly matched elements with respect to all the expected matchings. In other words, it measures how many correct matchings have been produced, as shown in equation 3.

$$Recall = \frac{|\{Retrieved\,Matches\} \cap \{Relevant\,Matches\}|}{|\{Relevent\,Matches\}|} \quad (3)$$

Finally, the *F-measure* combines both accuracy and recall in order to get an equally weighted average value of the measures. The corresponding formula is shown in equation 4.

$$F - Measure = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (4)$$

It is worth noting that all the considered evaluation measures produce a numerical result, ranging from 0 to 1, where 0 corresponds to the worst and the 1 to the best possible value.

7.3 Results

In this section, we discuss the obtained results with respect to *Precision*, *Accuracy* and *F-Measure*, first, and time performance, then.

7.3.1 Quality Performance Results

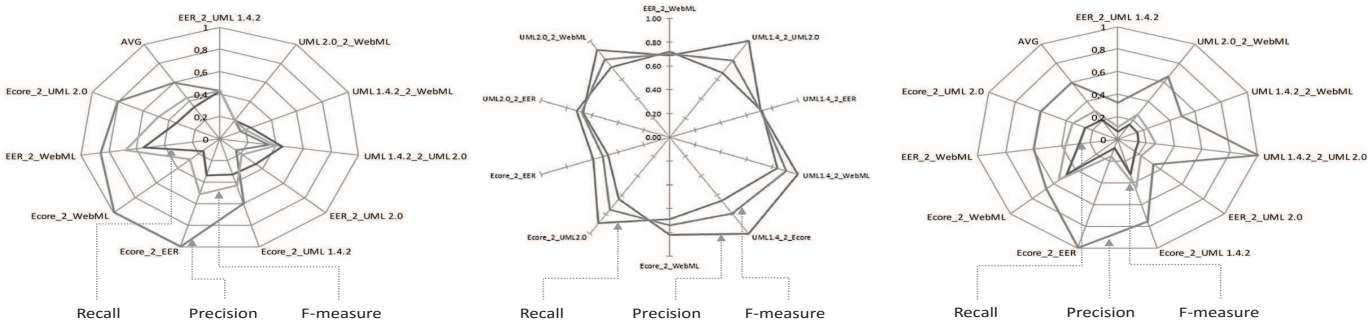
Figure 3 illustrates the results obtained with our proposed method, *GAMMA* and EMFCompare for the model exchange benchmark. Each axis of the glyph represents a matching task involving two metamodels from the initial set. The three metrics are represented using three quantitative variables, i.e. Precision, F-measure, and Recall.

Overall, our method provides the second best results with respect to *Precision*, *Recall* and *F-Measure*. The benchmark evaluation allows us to understand various characteristics about our solution, hence possible future improvements. First of all, the obtained results allow us to observe that our solution usually produces bigger number of matches than expected. In light of that using semantic approach, our extension matches more terms (i.e. *objects* and *item* by semantic extension and it does not by default implementation). Therefore *Precision* has in some cases a low value whereas considering *Recall* there is a significant improvement. However the *F-measure* has a better value than other tools analysed in [27].

Currently, the *GAMMA* [27] approach provides the best results in the state-of-the-art with respect to *Precision*, *Recall* and *F-Measure*. However, it is worth noting that *GAMMA* uses SBSE approaches to solve the matching problem, therefore the algorithm has to be initialised with a set of initial solutions which constitute the knowledge base for the computation.

7.3.2 Time Performance Results

Considering the main ideas driving the creation of EMFCompare, time performance is perhaps among the most important ones. Unlike for other approaches like [27], indeed, the tool has been de-



a) Semantic EMFCompare b) GAMMA c) EMFCompare

Figure 3: Comparison of the results related to the metamodels in the Model Exchange benchmark

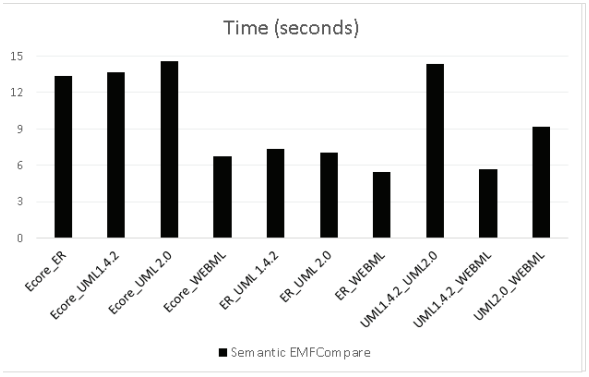


Figure 4: Execution time of Semantic EMFCompare

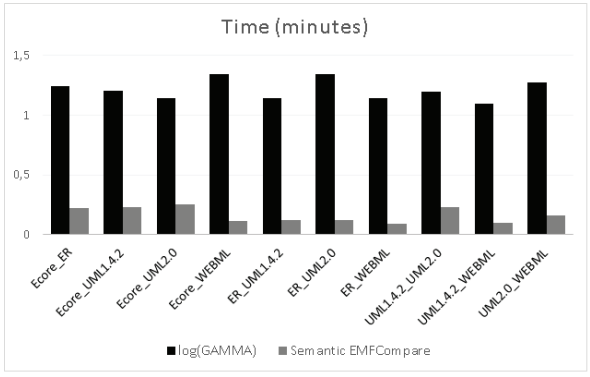


Figure 5: Comparison of execution time of GAMMA and Semantic EMFCompare

signed to provide a fast comparison algorithm, even at the price of showing some degradation with respect to the *Precision*, *Recall* and *F-Measure* of the produced results [6].

One of our main concerns was to introduce the semantic reasoning in the comparison process while keeping good time performances at the same time. As shown in figure 5, our extension successfully manages to perform a semantic reasoning while keeping acceptable time performances. In detail, our method works in terms of seconds as shown in 4, whereas for instance the solution proposed in [27] works in terms of minutes[2].

7.4 Discussion

The experience accumulated during the development of the proposed semantic extension of EMFCompare and its validation phase permits to draw some lessons learnt, which are detailed in this section.

Providing the comparison engine with a semantic reasoning can be done in a lightweight manner, as illustrated throughout this paper. However, we have noticed that an increasing matching power (thanks to the semantic reasoning) often comes to the price of an increasing imprecision, and in particular to a growing amount of false-positives or false-negatives that have to be traded-off. In fact, on the one hand the semantic reasoning is able to detect much more relationships between terms than a simple string comparison. On the other hand, the decomposition of entities' names not always produce terms known in the ontology exploited for the semantic reasoning (in our case WordNet), thus decreasing the similarity value. We tried to deal with this problem by tuning similarity thresholds and extending the information about the context in which the pair of elements is compared (i.e., including elements' owners and relationships). Nonetheless we feel that a more extensive experimentation is needed in order to achieve more reliable performances.

The issues mentioned above become very evident when comparing metamodels, since they are not expected to represent semantically related information, nor to contain concepts used in the usual English language. In this scenario, setting "looser thresholds" causes a lot of false-positives, while setting "tighter thresholds" remarkably reduces the number of matches. Therefore, quite surprisingly a pure syntactical comparison like the one computed by EMFCompare is able to provide on average more precise results than a differencing algorithm including some semantic reasoning. In this respect, even if comparing metamodels is a threat to validity, it is a factor that negatively affects the performances of the proposed extension. Therefore, we believe that our extension should be tested in the context of model comparison, given the higher probability of having semantically close names for the compared entities. Furthermore, we feel that the selection of the dictionary being used with respect to the kind of models to be matched plays a key role.

As a side remark, during our tests we have experimented the

[2]Please note that [27] comparison performances are rendered in a logarithmic scale for the sake of space.

lack of suitable mechanisms to generate differencing test cases. In fact, in order to validate the performances of our (and also other existing) differencing algorithm we would need an engine able to automatically produce different versions of a model and the corresponding expected comparison results.The current practice is to manually compare models and specify their differences, however this approach becomes quickly unfeasible with the increasing size of the input models. Alternatively, there exist tools and languages to generate sets of large models like Ecore Mutator[3] and Wodel [32] able to produce and track model manipulations (or mutations), however they typically work only on syntactical modifications. In this respect, there is the need to extend the mutation engine with the appropriate awareness of semantics variability, taking into account that there is always the risk of introducing some bias in the expected differencing results.

8. RELATED WORKS

Despite model versioning has been recognised as a critical feature for the successful adoption of MDE, performing model comparison with acceptable performances, both time- and precision-wise, is still an open research problem. In this respect, there exists a large body of literature discussing solutions to model comparison. A complete discussion of the existing literature goes far beyond the scope of this work, the interested reader can refer to [17, 27] as initial papers from which explore the field further. For the purpose of this paper, it is relevant to mention that in general approaches can deal with syntax or semantic matching. In both cases the comparison can be enriched by structural reasoning that helps in enhancing the precision of the matches [5]. Therefore, the extension proposed in this work can be considered as semantic matching with structural reasoning.

Other researchers have proposed semantic matching algorithms, like [33, 34, 35], just to mention a few. These techniques share the general approach of translating the syntax in a corresponding semantic domain, in which the matching is performed. Notably, both [34] and [35] rely on the behavioural semantics related to the compared models, while [33] proposes to translate the compared (meta-)models into corresponding ontologies and use ontology comparison algorithms. Our approach is closer to the last mentioned, since it exploits WordNet database and the linguistic relationships between terms to identify the possible semantic matches between the compared models. However, we do not consider the different methods as mutually exclusive, but rather potentially contributing to a more precise matching result. The open research problem is to provide empirical/formal foundations on how to appropriately combine the different techniques. As mentioned in this work, already combining syntactical and ontological matching is not straightforward in terms of the choice of the similarity thresholds, the order of the matching algorithms executions, and so forth.

The contribution by Kessentini et al. [27] is different from the previous ones since it is based on a search-based solution. Even if the precision of the matching can reach very good results, the unavoidable learning phase and the computation time can represent usability barriers of this and other similar solutions.

From a broader perspective, there exists a corpus of literature devoted to the problem of semantic clustering code to automatically extract trace links with requirements, documentation, design models, and more in general for information retrieval [36]. These techniques explore a set of artefacts, independently, with the goal of identifying clusters (also called topics), their characteristics (well-contained, cross-cutting, etc.), and the relationships among them. Then, the clusters in different documents are compared to identify matches [37]. Again, these approaches should not be conceived as mutually exclusive with respect to the exploitation of the semantic differencing illustrated in our proposal. On the contrary, they can constitute a useful characterisation of models for enhancing the precision of the matching algorithm.

[3]https://code.google.com/archive/a/eclipselabs.org/p/ecore-mutator

9. CONCLUSIONS AND FUTURE WORKS

In this paper, we presented an extension of the EMFCompare matching algorithm, which integrates the use of ontological information in order to calculate the similarity among two given model elements. The proposed solution represents an initial step towards the integration of semantic reasoning in the EMFCompare platform. Once presented our method, as well as the background notions it relies on, we also presented the results obtained from its application on the model exchange benchmark, which has been borrowed from [27]. Our solution does not provide the best results with respect to precision, recall and F-measure. However, the benchmark evaluation gives us useful information for possible future improvements. Moreover, we have successfully achieved to maintain fast time performance in our extension, therefore respecting one of the most important principles behind EMFCompare.

In the near future, we plan to further extend the semantic reasoning until completely cover the whole match engine. The current heuristics used in EMFCompare, indeed, are only reasonable when considering different versions of the same models, whereas tend to create inefficiencies whenever applied to models conforming to different metamodels. A possible solution to this inefficiency would be to allow the user to integrate an ontological specification of the differencing context, hence exploiting the consequent evaluations on its content.

Furthermore, although WordNet has been used in this paper, we have already defined our solution in order to be easily adaptable to different kind of ontological specifications. However, we plan to make our solution even more general in order to completely separate the ontological reasoning from its source description. Finally, it would be interesting to investigate the application of machine learning techniques in this context.

10. REFERENCES

[1] J. Bézivin, "On the unification power of models," *Software and Systems Modeling*, vol. 4, no. 2, pp. 171–188, 2005.

[2] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25–31, Feb. 2006.

[3] B. Selic, "The pragmatics of model-driven development," *IEEE Software*, vol. 20, no. 5, pp. 19–25, 2003.

[4] Z. Xing and E. Stroulia, "Umldiff: An algorithm for object-oriented design differencing," in *Procs of the 20th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE 2015)*, ser. ASE '05. New York, NY, USA: ACM, 2005, pp. 54–65.

[5] Y. Lin, J. Gray, and F. Jouault, "DSMDiff: a differentiation tool for domain-specific models," *European Journal of Information Systems*, vol. 16, no. 4, pp. 349–361, 2007. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00483464

[6] C. Brun and A. Pierantonio, "Model differences in the eclipse modeling framework," *The European Journal for the Informatics Professional*, April-May 2008.

[7] T. Mens, "A state-of-the-art survey on software merging," *IEEE Trans. Softw. Eng.*, vol. 28, no. 5, pp. 449–462, May 2002.

[8] R. C. Read and D. G. Corneil, "The graph isomorphism disease," *J. Graph Theory*, vol. 1, no. 4, pp. 339–363, 1977.

[9] D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige, "Different models for model matching: An analysis of approaches to support model differencing," in *Procs. of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, ser. CVSM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1 – 6.

[10] P. Oram, "Wordnet: An electronic lexical database. christiane fellbaum (ed.). cambridge, ma: Mit press, 1998. pp. 423." *Applied Psycholinguistics*, vol. 22, pp. 131–134, 3 2001.

[11] M. Alanen and I. Porres, *Difference and union of models*. Springer, 2003.

[12] P. Farail, P. Gaufillet, A. Canals, C. Le Camus, D. Sciamma, P. Michel, X. Crégut, and M. Pantel, "The topcased project: a toolkit in open source for critical aeronautic systems design," *Embedded Real Time Software (ERTS)*, vol. 781, pp. 54–59, 2006.

[13] A. Toulmé and I. Inc, "Presentation of emf compare utility," in *Eclipse Modeling Symposium*, 2006, pp. 1–8.

[14] R. Reddy, R. France, S. Ghosh, F. Fleurey, and B. Baudry, "Model composition-a signature-based approach," in *Aspect Oriented Modeling (AOM) Workshop*, 2005.

[15] C. Treude, S. Berlik, S. Wenzel, and U. Kelter, "Difference computation of large models," in *Procs. of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2007, pp. 295–304.

[16] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave, "Matching and merging of statecharts specifications," in *Procs. of the 29th Int. Conf. on Software Engineering (ICSE 2007)*. IEEE Computer Society, 2007, pp. 54–64.

[17] D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige, "Different models for model matching: An analysis of approaches to support model differencing," in *Procs of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*. IEEE Computer Society, 2009, pp. 1–6.

[18] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1091–1095, Jun. 2007.

[19] L. Meng, R. Huang, and J. Gu, "A review of semantic similarity measures in wordnet." *International Journal of Hybrid Information Technology*, vol. 6, no. 1, 2013.

[20] S. Patwardhan, S. Banerjee, and T. Pedersen, "Using measures of semantic relatedness for word sense disambiguation," in *Procs of the 4th Int. Conf. on Computational Linguistics and Intelligent Text Processing*, ser. CICLing'03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 241–257.

[21] A. Budanitsky and G. Hirst, "Evaluating wordnet-based measures of lexical semantic relatedness," *Comput. Linguist.*, vol. 32, no. 1, pp. 13–47, Mar. 2006.

[22] D. Lin, "An information-theoretic definition of similarity," in *Procs of the 15th Int. Conf. on Machine Learning*, ser. ICML '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 296–304.

[23] N. Seco, T. Veale, and J. Hayes, "An intrinsic information content metric for semantic similarity in wordnet," 2004.

[24] M. A. Rodríguez and M. J. Egenhofer, "Determining semantic similarity among entity classes from different ontologies." *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 2, pp. 442–456, 2003.

[25] Z. Zhou, Y. Wang, and J. Gu, "New model of semantic similarity measuring in wordnet," in *3rd Int. Conf. on Intelligent System and Knowledge Engineering, (ISKE 2008)*, vol. 1, Nov 2008, pp. 256–261.

[26] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Procs of the 2003 Conf. of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, ser. NAACL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 173–180.

[27] M. Kessentini, A. Ouni, P. Langer, M. Wimmer, and S. Bechikh, "Search-based metamodel matching with structural and syntactic measures," *J. Syst. Softw.*, vol. 97, no. C, pp. 1–14, Oct. 2014.

[28] J. Euzenat, "An api for ontology alignment," in *The Semantic Web – ISWC 2004*, ser. Lecture Notes in Computer Science, S. McIlraith, D. Plexousakis, and F. van Harmelen, Eds. Springer Berlin Heidelberg, 2004, vol. 3298, pp. 698–712.

[29] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm, "Schema and ontology matching with coma++," in *Procs of the 2005 ACM SIGMOD Int. Conf. on Management of Data*, ser. SIGMOD '05. New York, NY, USA: ACM, 2005, pp. 906–908.

[30] Y. Kalfoglou, B. Hu, D. Reynolds, and N. Shadbolt, "Capturing, representing and operationalising semantic integration (crosi) project - final report," University of Southampton, Technical Report, October 2005.

[31] M. Ehrig, "Foam - framework for ontology alignment and mapping; results of the ontology alignment initiative," in *Procs. of the Workshop on Integrating Ontologies. Volume 156., CEUR-WS.org (2005) 72–76*, 2005, pp. 72 – 76.

[32] P. Gómez-Abajo, E. Guerra, and J. de Lara, "Wodel: a domain-specific language for model mutation," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM, 2016, pp. 1968–1973.

[33] G. Kappel, H. Kargl, G. Kramler, A. Schauerhuber, M. Seidl, M. Strommer, and M. Wimmer, "Matching metamodels with semantic systems - an experience report," in *Datenbanksysteme in Business, Technologie und Web (BTW 2007), Workshop Proceedings, Aachen, Germany*, 2007, pp. 38–52.

[34] P. Langer, T. Mayerhofer, and G. Kappel, *Semantic Model Differencing Utilizing Behavioral Semantics Specifications*. Springer International Publishing, 2014, pp. 116–132.

[35] S. Maoz, J. O. Ringert, and B. Rumpe, "Summarizing semantic model differences," *CoRR*, vol. abs/1409.2307, 2014. [Online]. Available: http://arxiv.org/abs/1409.2307

[36] A. Kuhn, S. Ducasse, and T. Gîrba, "Semantic clustering: Identifying topics in source code," *Information and Software Technology*, vol. 49, no. 3, pp. 230 – 243, 2007, 12th Working Conference on Reverse Engineering.

[37] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, Oct 2002.