Towards a Hybrid Approach to Measure Similarity Between UML Models

Lucian José Gonçales
PPGCA, Universidade do Vale do Rio dos Sinos (UNISINOS)
São Leopoldo, RS
lucianj@edu.unisinos.br

Kleinner Farias
PPGCA, Universidade do Vale do Rio dos Sinos (UNISINOS)
São Leopoldo, RS
kleinnerfarias@unisinos.br

Vinicius Bischoff
PPGCA, Universidade do Vale do Rio dos Sinos (UNISINOS)
São Leopoldo, RS
viniciusbischof@edu.unisinos.br

ABSTRACT

Several approaches to measure similarity between UML models have been proposed in recent years. However, they usually fall short of what was expected in terms of precision and sensitivity. Consequently, software developers end up using imprecise, similarity-measuring approaches to figure out how similar design models of fast-changing information systems are. This article proposes UMLSim, which is a hybrid approach to measure similarity between UML models. It brings an innovative approach by using multiple criteria to quantify how UML models are similar, including semantic, syntactic, structural, and design criteria. A case study was conducted to compare the UMLSim with five state-of-the-art approaches through six evaluation scenarios, in which the similarity between realistic UML models was computed. Our results, supported by empirical evidence, show that, on average, the UMLSim presented high values for precision (0.93), recall (0.63) and f-measure (0.67) metrics, excelling the state-of-the-art approaches. The empirical knowledge and insights that are produced may serve as a starting point for future works. The results are encouraging and show the potential for using UMLSim in real-world settings.

CCS CONCEPTS

• **Software and its engineering → Design languages**; *Software configuration management and version control systems*;

KEYWORDS

Model Similarity , Case Study , Model Comparison Technique , Unified Modelling Language , Information Systems

ACM Reference Format:
Lucian José Gonçales, Kleinner Farias, and Vinicius Bischoff. 2019. Towards a Hybrid Approach to Measure Similarity Between UML Models. In *XV Brazilian Symposium on Information Systems (SBSI'19), May 20–24, 2019, Aracaju, Brazil.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3330204.3330226

SBSI'19, May 20–24, 2019, Aracaju, Brazil

ACM ISBN 978-1-4503-7237-4/19/05…$15.00
https://doi.org/10.1145/3330204.3330226

1 INTRODUCTION

Several approaches to measure similarity between UML models have been proposed in recent years [3, 9, 13]. WebDiff [17] and MADMatch [12] would be examples of these approaches. Software developers make use of these approaches to understand how similar UML models are, or even to identify their commonalities and differences. In distributed software development, for example, separate teams may change UML models in parallel. This allows developers to work on specific parts of UML models that are relevant to them. Before consolidating the changes made, they need to measure the similarities between the versions of the UML models created in parallel. The term *model similarity* can be briefly defined as a measure of correspondences between two models, M_A and M_B. This similarity is obtained after comparing the different aspects of M_A and M_B, including their semantic, syntactic, structural, and design ones.

We have learned from empirical studies [13, 14] that computing similarity is still considered a highly error-prone and time-consuming task. Even worse, the current similarity-measuring approaches usually fall short of what was expected in terms of precision and sensitivity [9, 13–15]. Consequently, software developers end up using imprecise approaches to figure out how similar UML models of fast-changing information systemts are. In practice, similarity tends to be computed subjectively by experts, giving rise to conflicting notions of similarity [5, 7]. Moreover, most approaches [10, 12, 17] do not take into account multiple aspects of the UML models; rather, they have focused on comparing models following a traditional but imprecise match-by-name approach [10, 17].

This article proposes UMLSim, which is a hybrid approach to measure similarity between UML models. It brings an innovative approach by using multiple criteria to quantify how UML models are similar, including semantic, syntactic, structural, and design criteria. A case study was conducted to compare the UMLSim with five state-of-the-art approaches through six evaluation scenarios, in which the similarity between realistic UML models was computed. Software developers benefit from using the UMLSim typically when executing maintenance tasks of UML models like versioning of UML models, pinpointing structural correspondences between UML models. Our results, supported by empirical evidence, show that, on average, the UMLSim presented high values for precision (0.93), recall (0.63) and f-measure (0.67) metrics, excelling the state-of-the-art approaches. The empirical knowledge and insights that are produced may serve as a starting point for future works. The results were encouraging and show the potential for using UMLSim in real-world settings.

The remainder of this article is organized as follows. Section 2 presents the main concepts used in this research. Section 3 presents the related work. Section 4 presents the proposed approach. Section 5 presents the evaluation of the proposed approach. Finally, Section 6 presents some concluding remarks and future works.

2 BACKGROUND

This chapter contains a brief description of the basic concepts involved in this work.

2.1 Similarity of Software Design Models

The similarity of software design models consists of an activity aimed at establishing a degree of similarity between elements of input models, M_A and M_B. For this, similarity approaches seek to find commonalities and differences between each element of the input models M_A and M_B, so that a degree of similarity between M_A and M_B can be computed. This degree of similarity varies from 0 to 1. A threshold like 0.7 is defined to help developers to make decisions regarding whether the model elements are similar or not. Similarity equal to or equal to 0.7 means that model elements are similar. Therefore, M_A and M_B elements are equivalent if their similarity is greater or equal to a previously defined threshold.

The M_A and M_B elements can be compared considering different aspects inherent to UML models, including syntactic, semantic, structural, and design ones. Heuristics to compute similarity can consider such aspects to produce more precise similarity measures. The syntactic aspect assigns a degree of similarity to the difference between element labels, and meta-model properties, while the semantic aspect calculates how similar the meanings of the models are. The structural aspect evaluates the hierarchical positions of the elements, while the design aspect evaluates similarity from a quantitative perspective of architectural issues, such as coupling, circular references, the quantity of attributes, operations, and relationships

2.2 Unified Model Language

The Unified Model Language (UML) was created from the unification of three visual notations of object-oriented projects: the Booch method, Object Modeling Technique (OMT), and Object-Oriented Software Engineering (OOSE) [18]. Since then, due to its wide adoption of the industry in communication support and understanding of design aspects of software systems [7], industry considered UML the *de facto* lingua franca of software development. UML has 14 models to represent the behavioral and static aspects of the diagrams [18]. Among them the class diagram is the most commonly used diagram to represent aspects of system design and implementation [7].

The class diagram visually represents the structure of a system's functionality, such as the attributes and methods it contains, and how these classes relate. Figure 1 presents a class diagram in the standard established by the UML, and the numbering of the following text corresponds to the identifiers contained in that figure. Next, some of the main components of this diagram is briefly described: (1) The entity class is composed of attributes and methods; and (2) The inheritance relationship between the *Chart* and *Circle* classes. It is a type of UML relationship that indicates a specific class (specialization) inherits the attributes and methods of a general class (generalization).

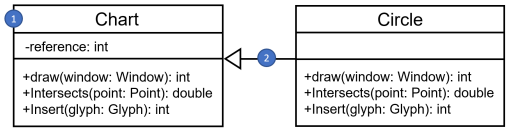

Figure 1: Example of a UML Class Diagram.

3 RELATED WORKS

This section details and makes a comparative analysis of the related works.

3.1 Analysis of the related works

Before conducting this research, we produced a search string [1], and used it on search engines such as ACM Digital Library[2] and Google Scholar[3] to find and retrieve related work. In total, 4 relevant articles and a tool were selected, which are described and analyzed as follows.

MadMatch [12]. The authors highlight that measuring the similarity of software design models is a frequent task for evolving and maintaining systems, but still lacks effective techniques. In this sense, they proposed MadMatch, which is an approach to measure similarities between software design models. This approach, measures the similarity based on syntactic, semantic, and structural aspects. Some contrasting point was identified in relation to this work. First, they do not consider evaluating the design aspect. Second, the tools were not evaluated on a realistic scenario. Moreover, the evaluation did not consider the comparison with another state-of-the-art tool.

MoCoto [6]. This study reports that the rigidity and inflexibility affect the quality of the techniques ecosystem for managing the integration of models, including the similarity techniques. In this sense, the authors propose MoCoto, a technique for integrating software design models. Despite this technique is focus on integrating design models, it executes a similarity calculation step, i.e., similarly we proposed in our work. This MoCoto module calculates the similarity based on semantic, and syntactic aspects. However, some differences were found in relation to the proposed work. First, they concern on improving the correctness of integration of software design models, instead the precision of similarity techniques. Second, their similarity measuring step considers only the semantic, and syntactic aspects. Besides these aspects, the proposed technique also considers the structural and design operators. Third, unlike our study they did not conduct any experimental study to evaluate the effectiveness of related approaches.

WebDiff [17]. The authors are concerned on the lack of approaches that properly calculates the similarity of software design models. For this, they proposed WebDiff, a tool for calculating

[1] ((UML OR Unified Modelling Language) AND (Comparison OR Similarity OR Matching OR differencing) OR (Diagrams OR Model OR Design))
[2] https://dl.acm.org/
[3] https://scholar.google.com.br/

model differences based on a match-by-name approach. This work presents some differences in relation to our proposed work. First, WebDiff calculates the similarity based on the elements name, i.e., it is categorized as a syntactic similarity aspect. Instead, our work executes the comparison also based on semantic, structural, and design aspects. Second, this study did not conduct any experiment or comparative analysis in relation to the proposed state-of-the art tools. Third, WebDiff focused on producing differences of software models and does not produce a general similarity value that can serve as input to many software development activities.

Astah [1]. Astah was concerned on providing a market solution to support the modeling of UML diagram's ecosystem. Furthermore, this tool provides a feature for comparing software design models. This feature aims to differentiate the design models within the context of software evolution. However, we identified some differences in relation to our approach. First, the similarity of the diagram is id-based instead of using heuristics similarity, and therefore, none of the aspects of the diagram are considered to establish equivalence relations. Second, we did not find any work reporting the precision of Astah comparison features, neither conducting an analysis of Astah in relation to other tools.

EMFCompare [2]. The authors were concerned over the need to maintain a history of evolution of software models where collaborators could view the changes made during subsequent versions. Therefore, the authors proposed the EMFCompare, an approach to compare software design models and thereby identify changes during software evolution. EMFCompare considers the syntactic and semantic aspects of the models to measure the degree of similarity. In contrast, the proposed technique also considers the structural and design aspects.

3.2 Comparative analysis of the works

This section contrasts the proposed extension with the previously analyzed studies. This comparison, based on comparison criteria (C), serves to identify some similarities and differences. The comparison criteria are presented below:

(1) **Main contribution (C1)**: Studies that have as main contribution a similarity approach for software design models.
(2) **Proposed approach (C2)**: Studies that introduce a new approach that deal with research topics related to similarity of software design models, such as, model matching, model merging, or model evolution.
(3) **Similarity aspects (C3)**: Studies that evaluates the similarity between two input models based on semantic, syntactic, structural, and design aspects.
(4) **Experimental study (C4)**: Studies that evaluated the proposed approach through empirical studies, such as survey, controlled experiment, quasi-experiment or case study.
(5) **Context (C5)**: Studies that have been performed evaluations with rich-semantic artifacts in academia, i.e., those nearly used and produced on industry context.
(6) **Study Variables (C6)**: Studies that analysed the precision, accuracy, and recall of the correctness of similarity approaches.
(7) **Comparative Analysis (C7)**: Studies that compares their proposed techniques to another state-of-arts approaches.

Table 1 presents the comparison considering these criteria. It is observed that, only the proposed work fully meets the defined criteria, highlighting the contribution and the differential of this work.

Table 1: Comparative analysis of related works.

	Comparison Criteria						
Related Works	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**
UMLSim	●	●	●	●	●	●	●
MADMatch [12]	●	●	◐	●	○	●	○
MoCoto	◐	●	◐	○	○	○	○
WebDiff [17]	◐	◐	◐	○	○	○	○
Astah [1]	◐	◐	○	○	○	○	○
EMFCompare [2]	◐	◐	◐	○	○	○	○

Legend
● Meets Fully ○ Does not meet
◐ Meets partially ⊘ Not Applicable

Research opportunity. To sum up, none of the proposed tools considered a hybrid approach to evaluate software design models. Instead, they have operators that can not be used independently turning the techniques rigid, and prone to errors due to inflexibility to adapt to different contexts of domains. For this, this work proposed an approach that implement four independent operators that turn this technique hybrid and multicriteria. Furthermore, instead of limiting the evaluation to only semantic or syntactic criteria, the proposed tool is the first to evaluate structural, and design aspects.

4 THE PROPOSED APPROACH

This section introduces UMLSim, which is a hybrid approach for measuring the similarity between UML models. The UMLSim differential comprises considering different but complementary aspects of UML models like semantic, syntactic, structural, and design—where the term *hybrid* comes from.

4.1 Overview of the UMLSim Process

Figure 2 presents an overview of the adopted process, along with the proposed approach. This process brings together all operators seamlessly. These operators calculate the similarity between the input models, M_A and M_B, through a specific perspective. The similarity is a value from 0 to 1. Currently, UMLSim only supports UML class diagram since they are the most widely adopted in practice. Figure 3a shows an illustrative example of UML class diagram. Each step of the UMLSim process is described as follows:

Step 1: syntactic similarity. This step aims at computing the similarity between two input models, M_A and M_B, considering their syntactic aspect [16]. For this, the *syntactic operator* evaluates their elements following a match-by-name strategy. The syntactic similarity between pairs of M_A and M_B elements are stored in a $m \times n$ syntactic similarity matrix (SS), as shown in Figure 3b, where m (row) and n (column) are the number of M_A and M_B model elements, respectively.

Step 2: semantic similarity. This second step seeks to measure the semantic similarity [12] between pairs of M_A and M_B elements. We evaluate how similar are the meanings

of the terms used to name the M_A and M_B elements. In this sense, the *semantic operator* computes and stores the obtained results in a $m \times n$ semantic similarity matrix (SE), where m (row) and n (column) are the number of M_A and M_B model elements, respectively.

Step 3: structural similarity. This step seeks to discover how similar the structures of M_A and M_B are [12]. The *structural operator* analyses the influence of class neighbours. The *structural operator* stores the obtained results in a $m \times n$ structural similarity matrix (SN), where m (row) and n (column) are the number of M_A and M_B model elements, respectively.

Step 4: design similarity. This step aims at determining the similarity between M_A and M_B based on design metrics [21]. The *design operator* receives as M_A, M_B and design metrics as inputs. Examples of design metrics would be the number of elements on which this class depends, and the depth of the class in the inheritance hierarchy. The *design operator* runs a set of metrics and then stores the obtained results in a $m \times n$ design similarity matrix (SM), where m (row) and n (column) are the number of M_A and M_B model elements, respectively.

Step 5: general similarity. This last step is responsible for bringing together the syntactic, semantic, structural and design similarities so that a general similarity can be computed. Each previous operator outputs a similarity matrix. It has inputs the matrices produced in the privious steps, and weight (explained later). This step finishes producing a $m \times n$ *final similarity matrix*, a m (row) and n (column) are the number of M_A and M_B model elements, respectively. The elements of this matrix are denoted by $a_{i,j}$, which corresponds to a similarity value between an *i-element* of M_A and another *j-element* of M_B. The similarity matrix is complete when the last $a_{n,m}$ element is calculated.

The following section describes such similarity operators in more detail.

4.2 Similarity Operators

This section describes the four UMLSim Similarity Operators.

4.2.1 Syntactic Operator. The UML metamodel specification [18] puts some light about the definition of the syntax of UML models. Basically, UML models has a concrete and abstract syntax. The UML metamodel specification of each diagram defines the respective abstract syntax of them. The concrete syntax of UML models refer to the way diagrams are constructed, interchanged, and represented. This technique evaluates the concrete syntax of UML models because this technique does not focus on comparison of metamodels. The constructions and representations on concrete syntax refers to elements such as the attributes, methods, and classes. Therefore, the syntax similarity is based on how these constructs differ from a diagram to another.

This operator calculates the similarity of concrete syntax based on a match-by-name strategy [16]. Then, this technique analyses the similarity between the labels of model elements such as names of class, attribute, and method. In addition, UMLSim also considers other properties, such as whether elements are abstract, and their visibility. This way, the syntactic evaluation of UMLSim can consider different level of detail for similarity evaluation. The details of the input diagrams are known as granularity, or conflict units. In this technique, a granularity has three levels: coarse-grained, partial and fine-grained. In coarse-grained granularity, a syntactic analysis considers the names of classes, attributes, and methods. In the partial granularity, a partial amount of the properties of diagrams will be considered on the similarity. Finally, in fine-grained all properties of diagram elements are evaluated.

Equation 1 presents a formula of syntactic similarity. The calculation uses Lenvenshtein [22] to calculate the distance between two labels. Length is an operation that calculates the size of the label. A similarity between labels is given by Equation 1:

$$labelSimilarity(label_1, label_2) = \frac{levenshteinDistance(label_1, label_2)}{higherValue(length(label_1, label_2))} \quad (1)$$

As an example, the comparison between two classes of input models M_A and M_B, Line and Edge would result on 0.25 by Equation 1, indicating that Line has 25% similarity to Edge. This is because, the *levensteinDistance*(Line, Edge) = 3, and *length*(Line) = 4, *lenght*(Edge) = 4. Figure 4 shows this result on the syntactic similarity matrix.

4.2.2 Semantic Operator. UML superstructure [18] categorises the semantics regarding the systems context in two groups: (1) structural semantics, i.e., the semantic of static elements, and (2) behavioural semantics, i.e., the semantic of dynamic elements. The UMLSim considers the structural semantics of UML diagrams because it supports Class Diagrams. In practice, UMLSim evaluates whether the input models are similar in relation their domain of discourse. This means that structural terms must be evaluated according their terms proximity. The terms may be different syntactically, but similar according the context of the modelled system. For this, UMLSim provides a semantic operator which is responsible for evaluating the meaning of terms between input models. This similarity is based on a synonym dictionary. Developers and engineers must use this dictionary to link related terms. Developers must link terms that are equivalent inside the domain of a discourse.

Equation 2 presents how the semantic operator attributes similarity values between terms. The equation evaluates if the terms are synonyms. The terms are completely similar in the case they are synonyms, and not similar otherwise. As an example, the comparison between two classes of input models M_A and M_B, Line and Edge. The result of Equation 2 is 1, indicating these terms are 100% similar. This is because these terms are related as synonyms on the dictionary. Figure 4 shows this result on the semantic similarity matrix. In addition, this similarity indicates that both classes are equivalent in terms of meaning, i.e., they have the same purpose. However, this operator does not evaluate if both classes are equivalent regarding their neighbours. This could impact on the contextual similarity of both classes. For this, the structural similarity is important to be evaluated. In particular, the structure of classes surrounds both elements. For this, next section presents an operator to evaluate the similarity of the structures that surrounds these classes.

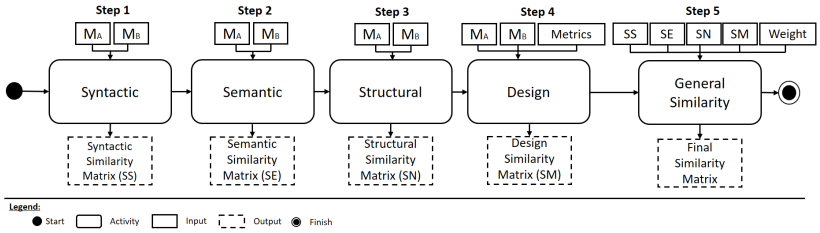

Figure 2: Overview of the UMLSim approach.

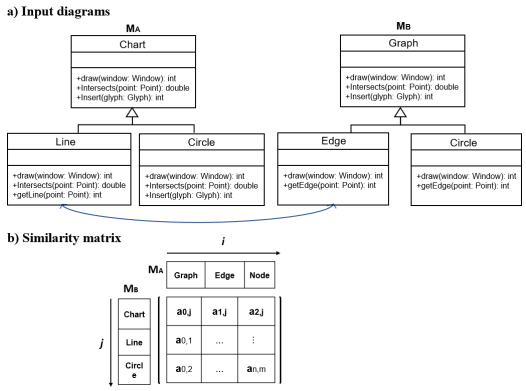

Figure 3: Input models M_A and M_B and their similarity matrix.

$$SynonimSym = \begin{cases} 1 & isSynonim(term_1, term_2) == true \\ 0 & isSynonim(term_1, term_2) == false \end{cases} \quad (2)$$

4.2.3 Structural Operator. The previous sections were limited to point two types of similarities: (1) identify the difference between names of the class structures, and (2) evaluates if names are different but make part of the same context. However, they fail to account in the similarity value when the elements have the same name but differ regarding their context. Then, they also fail in evaluating the structure of the diagrams in respect of objects surroundings. As solution to this is to evaluate the structural similarity [8][19][11]. However, they not converged in how to evaluate diagrams structure. In UMLSim, the structural similarity is to evaluate the direct impact neighbor has on compared elements. Thus, in this work classes are considered neighbor if there is at least an established relationship between them. This operator analyses the similarity of the diagram in relation to their neighbors. This operator assumes that an element of M_A has a correspondent element in M_B whether their neighbors are sufficiently similar.

Equation 3 presents how the operator evaluates the structural similarity. This equation means that given a $Class_i$ in M_A and a $Class_j$ in M_B, each respective neighbor of $Class_i$ (N_i) will be compared to each respective neighbor of $Class_j$ (N_j). The neighbors are nodes that have a relation in the Adjacency Matrix A. These neighbors are evaluated considering the semantic and syntactic criteria, as previously presented. The sum of the similarity values is divided by the value M. The M value corresponds to the total number of comparison operations between the neighbors of $Class_i$ and $Class_j$.

As an example, the structural comparison between "Line" and "Edge" class are SimEstrutural(Line, Edge) = 0.1, because their respective neighbors "Chart" and "Graph" have few syntactical similarities, and no semantic relation. Therefore, these classes are not equivalent based on the defined threshold of 0.7 similarity degree. These result are the structural similarity matrix shown in Figure 4.

$$NSim(M_A, M_B) = \frac{\sum_i^1 \sum_j^1 (sim(N_i, N_j))}{M} \quad (3)$$

4.2.4 Design Operator. Many design aspects can differentiate a model from another, such as a class concentrating attributes and methods, and the degradation of application layers. These design aspects are identified by a set of metrics, such as the number of attributes, methods, and interfaces. These simple metrics are usually applied in software engineering to support developers to take a quantitative view of the models. Thus, several failures are detected, which were difficult to perceive without these metrics, e.g., a presence of a God Class, i.e., a class that concentrates to much methods. Thus, the similarity of design metrics aims to compare how similar the input elements are in relation to these problems. Next, the MetricSim (metric similarity) operation (Equation 4) calculates the similarity between the elements the metric "n", i.e., their similarity in terms of God Methods, or their cohesion, and couple.

As an example, let "n" be the "number of methods" (NOM) as the metric being evaluated. Then, suppose that the NOM of a given class in M_A is 3, and the NOM of a given class in M_B is 8. Then, 3/8 = 37.5, i.e., they are 62.5% similar.

$$DesignSim(M_A, M_B) = \frac{\sum_n^1 [MetricSim_n(sim(element_A, element_B))]}{\sum_n^1 P_n} \quad (4)$$

4.2.5 General Operator. The general operator establishes the final degree similarity between the elements of software design. To generate the final similarity, the results of four comparison operators converges to a general similarity between the correspondent elements as shown in Figure 4. The user can assign a specific weight for each operator to prioritize which aspect has more relevance in relation to the abstraction level of the models. The weight can vary from 0 to 1 for each criterion. Thus, values close to zero indicate little relevance of the criterion in the evaluation of similarity, while values close to one indicate criteria that have much relevance within the evaluation of similarity. By default the weight is set to

1, i.e., all the aspects have the same relevance. Thus, the relevance of the structural and design aspect would decrease in the scenario where the compared models has a high abstraction level.

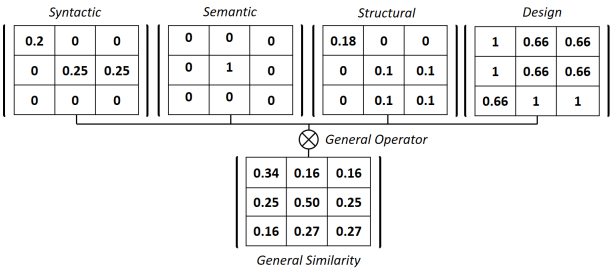

Figure 4: The general operator of the proposed approach.

Algorithm 1 presents the implementation of the general operator. They retrieve each similarity value from the respective matrix from Line 4 up to Line 7. Finally, it calculates a general similarity with their respective weight (Line 8).

Algorithm 1 General Similarity Operator

```
1: procedure GENERAL-OPERATOR(M_A, M_B, and weights of respective operators)
2:     for each element i in M_A do
3:         for each element j in M_B do
4:             SyntaticSimilarity ← SynctaticMatrix(i, j);
5:             SemanticSimilarity ← SemanticMatrix(i, j);
6:             StructuralSimilarity ← StructuralMatrix(i, j);
7:             DesignSimilarity ← DesignMatrix(i, j);
8:             GeneralSimilarity ← (P_S * SyntaticSimilarity + P_SE *
   SemanticSimilarity + P_E * StructuralSimilarity + P_D * DesignSimilarity)
   / (P_S + P_SE + P_E + P_D);
9:             GenSimMatrix[i][j] ← GeneralSimilarity;
10:        end for
11:    end for
12:    Return GenSimMatrix;
13: end procedure
```

5 EVALUATION

This section presents the evaluation of the UMLSim.

5.1 Scenarios

The UMLSim and related comparison techniques were evaluated in six different scenarios [4]. Each scenario corresponds to an evolution of model M_B from the model M_A. A developer were invited in order to changed and evolve the Model M_A, and then generated M_B. Thus, after the changes, aiming to evolve the software model he needs to reconcile the overlapping and different parts which are identified by comparison techniques. These scenarios are described in Table 2, and were derived from the source code of respective projects contained on GitHub repository. The scenarios are e-commerce, ATM locator, law firm, petrol station, pet system, and university system domains. We chose different types of domains for demonstrating the behaviour of proposed approach and state-of-the-art techniques in the various context of comparison.

Figure 5 shows an example of a scenario we used. It is an UML class diagram representing a system on the e-commerce domain. This application implements a sales system that allows the customer to login on the system, visualize the product, and make payment. Table 2 shows a detailed description of all scenarios.

[4]https://drive.google.com/open?id=1rJHBrrFQzRjPXksEhi9O6Kig9ajnhKU6

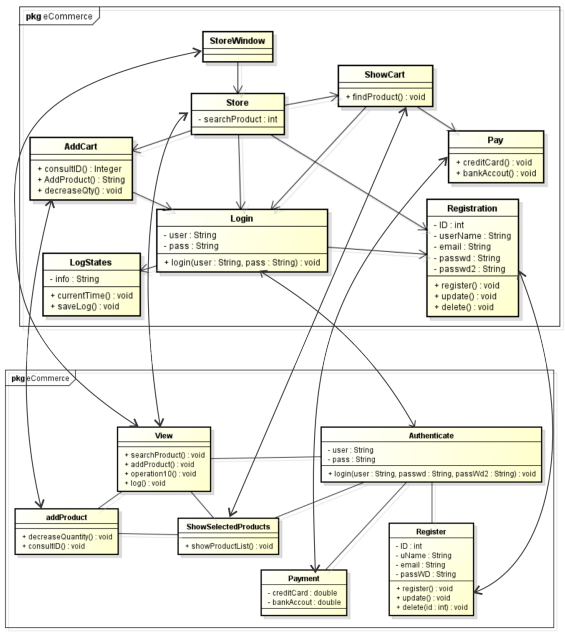

Figure 5: The bidirectional arrows represents matching elements between M_A and M_B

Table 2: Description of comparison scenarios.

Scenarios	M_A		M_B		# Equivalent Classes
	Elements	# Elements	Elements	# Elements	
1	Class	7	Class	9	7
	Attributes	19	Attributes	8	
	Methods	15	Methods	12	
2	Class	8	Class	6	6
	Attributes	8	Attributes	8	
	Methods	13	Methods	11	
3	Class	6	Class	4	4
	Attributes	17	Attributes	12	
	Methods	10	Methods	8	
4	Class	6	Class	6	5
	Attributes	15	Attributes	13	
	Methods	12	Methods	6	
5	Class	6	Class	5	6
	Attributes	7	Attributes	6	
	Methods	14	Methods	12	
6	Class	18	Class	6	5
	Attributes	29	Attributes	12	
	Methods	3	Methods	14	

5.2 Variables

The tool's precision was evaluated through the quantitative precision, recall, and f-measure variables [20]. The equation's results vary between 0 and 1. Values near 1 means that results are precise, and values close to 0 points fewer accurate results. In the context of this work, precision (Figure 6(a)) is defined as the number of classes that were correctly assigned as similar, divided by the set of all classes that should be equivalent.

Table 3: The obtained results related to the metrics of precision, recall and f-measure.

Scenarios	Metrics	EMF Compare	MAD Match	Astah	MoCoto	Web diff	UML Sim
eCommerce	Precision	0.4	0.27	0.85	0.41	0.08	1
	Recall	0.57	0.37	0.85	0.62	0.16	0.66
	F-measure	0.47	0.31	0.85	0.5	0.11	0.8
ATM	Precision	0.11	0.16	1	0.36	0.06	1
	Recall	0.25	0.28	1	0.57	0.14	1
	F-measure	0.16	0.21	1	0.44	0.09	1
Advocacy System	Precision	0.1	0.16	0.57	0.28	0.11	1
	Recall	0.2	0.14	0.66	0.33	0.25	0.8
	F-measure	0.13	0.15	0.61	0.3	0.15	0.88
Pet System	Precision	0.2	0.16	0.5	0.66	0.18	1
	Recall	0.33	0.33	0.66	0.66	0.33	0.33
	F-measure	0.25	0.22	0.57	0.66	0.23	0.5
Petrol System	Precision	0.22	0.1	0.37	0.57	0.18	1
	Recall	0.33	0.16	0.6	0.66	0.33	0.33
	F-measure	0.26	0.12	0.46	0.61	0.23	0.5
University System	Precision	0.04	0.11	0.27	0.22	0.08	0.6
	Recall	0.05	0.11	0.27	0.22	0.05	0.18
	F-measure	0.05	0.11	0.27	0.22	0.06	0.28
General Average	Precision	0.18	0.13	0.59	0.42	0.1	0.93
	Recall	0.29	0.21	0.67	0.51	0.2	0.63
	F-measure	0.22	0.16	0.63	0.45	0.13	0.67

In particular, the components of the precision (Figure 6(a)) are the true positive results (tp), i.e., the correct results, and the false positive results. The recall (Figure 6 (b)) indicates if the set of all possible correct answers was achieved. For this, it considers the true positive results (tp), i.e., the number of right matchings, and false negatives (fn) that are the results correctly pointed as false. Finally, we also evaluated the f-measure (Figure 6 (c)) of the comparison tools. The f-measure is the harmonic average between precision and recall. For this both values of precision and recall are considered in this equation.

Figure 6: Precision, Recall, and F-measure equations.

Precision (a) $precision = \frac{tp}{tp+fp}$ **Recall (b)** $recall = \frac{tp}{tp+fn}$ **F-Measure (c)** $f - measure = \frac{2.(precision*recall)}{(precision+recall)}$

5.3 Obtained Results

Figure 7 presents an overview of the results obtained from the current comparison tools produced through Kiviat diagrams. Each axis in the graph corresponds to the comparison between two input models, M_A and M_B in one of the comparison scenarios. In addition, three distinct colors represent the quantitative variables respectively. The blue line represents the precision. The red line represents the recall values. Finally, the green color represents the f-measure variable.

Results of the approach proposed in the general context of the scenarios. Overall, the results indicate that the precision of the proposed approach reached at least 60% in all scenarios. The precision value in the ATM scenario was also the largest, where the precision, recall, and f-measure variables reached 100%. This shows that the precision of the proposed approach reaches a considerable result in different scenarios.

Combination of recall and f-measure. The Astah has the highest score considering the combination of recall and f-measure (recall = 0.85 and f-measure = 0.85). However, Astah calculates the similarity based on the tracing of the objects identifier. This traceability is lost when someone removes some element from the diagram and inserts it again. This is a reason to justify why tools that implement a more sophisticated technique have lower scores than Astah on some scenarios, i.e., in the advocacy and university scenarios, MadMatch (recall = 0.14, and f-measure = 0.15), and WebDiff (recall = 0.05, and f-measure = 0.06) presented low scores.

General Average of Precision. The UMLSim tool obtained the highest mean of precision (Table 3), presenting an average of 93% in correctness in the equivalences between the components. The proposed tool achieved 100% of precision in the ATM, eCommerce, Advocacy System, Pet System, and Petrol System. As previously mentioned, the proposed tool mapped the equivalences with a precision of at least 60% in the University System scenario.

General Average of Recall and F-measure. Table 3 shows that the proposed tool obtained a relevant recall average (63%), as well as, the higher f-measure average (67%). Despite both f-measure and recall are higher in ATM scenario (100%), the determinant factor that had a negative impact on the general average is the result of recall and f-measure in the University system, i.e., 18% and 28% respectively. This means that the tool had precision in few results in this scenario. This is because the university scenario was designed with many-to-one equivalences, i.e., many class equivalent to a only one class. This is a problem that will be addressed in future research. Another interesting result is that the UMLSim had a recall lower than Astah, and MoCoTo measuring the similarity of PetSystem, PetrolSystem and University System scenarios. This is because the Astah is a tool that maintains an id-based traceability between elements. The recall were inferior than MoCoTo because the equal distribution of weights between the evaluated criteria may not be appropriated for these specific cases. Finally, the evalution was focused on scenarios of small size. Thus, evaluating the scalability between models that represents ecosystems of information systems is a future work [4].

6 CONCLUSIONS AND FUTURE WORK

This work proposed a hybrid approach and multi-criteria for comparison software model called UMLSim. It is composed by syntactic, semantic, structural, and design similarity operators. The results showed that the technique performed well among the different scenarios evaluated. Thus, the use of multicriteria is a way to improve the precision and recall of similarity between the models. The usage of these criteria improved the precision in the majority of the scenarios. For software industry, an hybrid approach is essential because it enables developers calibrate the comparison aspects according the abstraction level of the diagrams. This turns the similarity more precise, and the comparison process more adjustable to the type of models being used.

The main contributions of this work was: (1) a brief description of state-of-the-art tools produced on academia and industry about model comparison; (2) an approach for evaluating the similarities based on the gaps and shortcomings identified related to contemporary tools; (3) the generation of empirical knowledge on the effectiveness of contemporary model comparison techniques.

As future work, we intend to replicate this case study on the evaluation of related tools aiming at a more detailed analysis, e.g.,

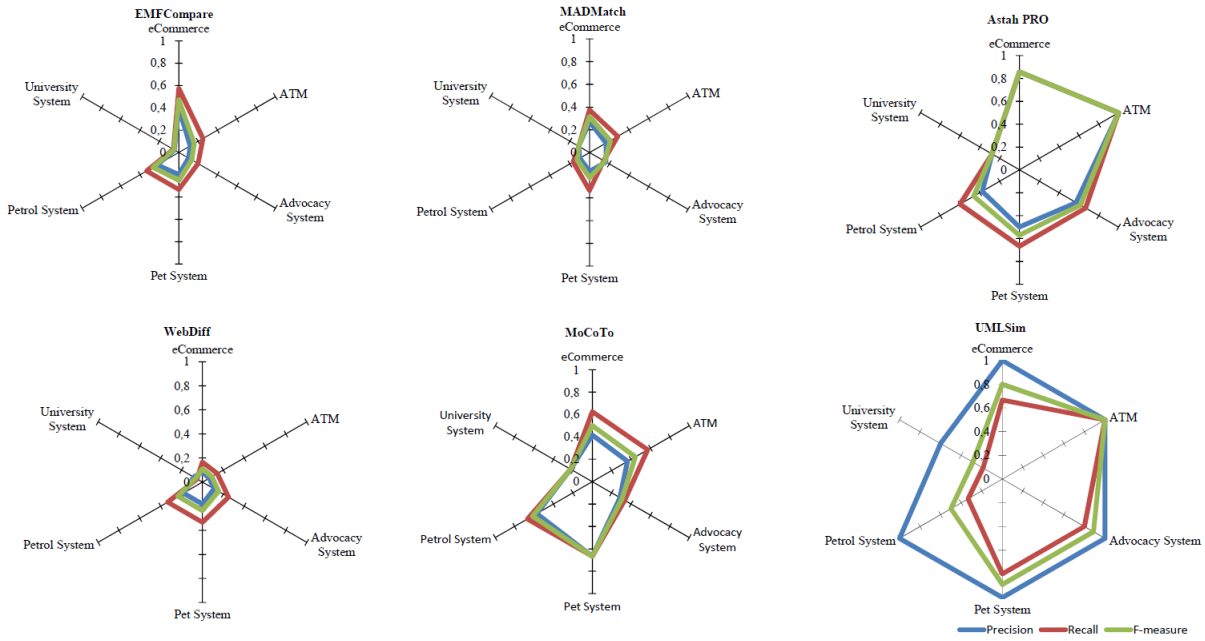

Figure 7: Results related to precision.

testing using models with greater size. In addition, we seek to add other similarity operators, which are able to apply concepts greedy search in the context of model similarity. We also intend to conduct an experiment on the invested effort to resolve conflicts generated by heuristic comparison techniques in relation to specification techniques.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior -Brasil (CAPES) - Finance Code 001, and the Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS).

REFERENCES

[1] Astah. 2018. http://www.astah.net/.
[2] Cédric Brun and Alfonso Pierantonio. 2008. Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional* 9, 2 (2008), 29–34.
[3] Hugo Bruneliere, Florent Marchand de Kerchove, Gwendal Daniel, and Jordi Cabot. 2018. Towards Scalable Model Views on Heterogeneous Model Resources. In *Proc. of the 21th ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems*. ACM, 334–344.
[4] Kattiana Constantino, Eduardo Figueiredo, Glauco Carneiro, and Raquel Minardi. 2016. Multiple View Interactive Environment to Analyze Software Product Line Tools. In *Proc. of the XII Brazilian Symposium on Information Systems*. 32.
[5] Hoa Khanh Dam, Alexander Egyed, Michael Winikoff, Alexander Reder, and Roberto E Lopez-Herrejon. 2016. Consistent merging of model versions. *Journal of Systems and Software* 112 (2016), 137–155.
[6] Kleinner Farias, Alessandro Garcia, Jon Whittle, Christina von Flach Garcia Chavez, and Carlos Lucena. 2015. Evaluating the effort of composing design models: a controlled experiment. *Software & Systems Modeling* 14, 4 (2015), 1349–1365.
[7] Rodi Jolak, Boban Vesin, and Michel RV Chaudron. 2017. OctoUML: an environment for exploratory and collaborative software design. In *39th International Conference on Software Engineering. ICSE*, Vol. 17.
[8] Marouane Kessentini, Ali Ouni, Philip Langer, Manuel Wimmer, and Slim Bechikh. 2014. Search-based metamodel matching with structural and syntactic measures. *Journal of Systems and Software* 97 (2014), 1–14.
[9] Alexander Knapp and Till Mossakowski. 2018. Multi-view Consistency in UML: A Survey. In *Graph Transformation, Specifications, and Nets*. Springer, 37–60.
[10] Dimitrios Kolovos, Louis Rose, Richard Paige, and Antonio Garcia-Dominguez. 2018. *The Epsilon Book*.
[11] Danai Koutra, Neil Shah, Joshua T Vogelstein, Brian Gallagher, and Christos Faloutsos. 2016. DeltaCon: principled massive-graph similarity function with attribution. *ACM Trans. on Knowledge Discovery from Data (TKDD)* 10, 3 (2016), 28.
[12] S. Kpodjedo, F. Ricca, P. Galinier, G. Antoniol, and Y. G. Guéhéneuc. 2013. MADMatch: Many-to-Many Approximate Diagram Matching for Design Comparison. *IEEE Trans. on Software Engineering* 39, 8 (Aug 2013), 1090–1111.
[13] Kristóf Marussy, Oszkár Semeráth, and Dániel Varró. 2018. Incremental View Model Synchronization Using Partial Models. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, 323–333.
[14] Gleiph Ghiotto Lima Menezes, Leonardo Gresta Paulino Murta, Marcio Oliveira Barros, and Andre Van Der Hoek. 2018. On the Nature of Merge Conflicts: a Study of 2,731 Open Source Java Projects Hosted by GitHub. *IEEE Transactions on Software Engineering* (2018).
[15] Johnatan Oliveira, Eduardo Fernandes, Maurício Souza, and Eduardo Figueiredo. 2017. A method based on naming similarity to identify reuse opportunities. *iSys-Revista Brasileira de Sistemas de Informação* 10, 1 (2017), 99–121.
[16] Kleinner SF Oliveira, Karin Koogan Breitman, and Toacy Cavalcante de Oliveira. 2009. A Flexible Strategy-Based Model Comparison Approach: Bridging the Syntactic and Semantic Gap. *J. UCS* 15, 11 (2009), 2225–2253.
[17] N. Tsantalis, N. Negara, and E. Stroulia. 2011. Webdiff: A generic differencing service for software artifacts. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. 586–589.
[18] OMG UML. 2011. *2.4.1 superstructure specification*. Technical Report. document formal/2011-08-06. Technical report, OMG.
[19] Konrad Voigt and Thomas Heinze. 2010. Metamodel matching based on planar graph edit distance. In *International Conference on Theory and Practice of Model Transformations*. Springer, 245–259.
[20] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.
[21] J Wust. 2019. SDMetrics: The software design metrics tool for UML. https://www.sdmetrics.com/.
[22] Li Yujian and Liu Bo. 2007. A normalized Levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, 6 (2007), 1091–1095.