Toward an Architecture for Comparing UML Design Models

Lucian José Gonçales*, Kleinner Farias, Vinícius Bishoff, Matheus Segalotto
Interdisciplinary Postgraduate Program in Applied Computing (PIPCA), University of Vale do Rio dos Sinos (UNISINOS), São Leopoldo, Rio Grande do Sul, Brazil.

* Corresponding author. Email: lucianjosegoncales@gmail.com
Manuscript submitted January 10, 2017; accepted March 8, 2017.
doi: 10.17706/jsw.12.7.559-569

Abstract: Academia and industry are increasingly concerned with producing general-purpose model comparison techniques to support many software engineering activities, e.g., clone detection or model composition. However, the current methods fail to provide flexible and reusable architectures, a comprehensive understanding of the critical composition activities, and guidelines about how developers can use and extend them. These limitations are one of the reasons why state-of-the-art techniques are often unable to aid the development of new comparison tools. To overcome these shortcomings, this paper, therefore, proposes a flexible, component-based architecture for aiding the development of comparison techniques. Moreover, an intelligible comparison workflow is proposed to support developers to improve the understanding of significant comparison activities and their relationships.

Keywords: Model comparison tool, model driven architecture, software architecture, unified modeling language.

1. Introduction

The comparison of software design models plays a central role in many software engineering activities, e.g., identifying clone models to assure the right authorship [1], detecting architectural patterns in design models [2], enhancing the retrieval of UML diagrams [3], computing the similarity between overlapping parts of design models to merge them [4]. For example, in the context of distributed software development, virtual teams might use comparison techniques to grasp how similar evolving design models are, or even helping to reconcile conflicting parts of design models that have been overly changed in parallel.

The term *comparison of design models* can be briefly defined as being the process of identifying equivalence relations between the content of design models. One way of quantifying the degree of equivalence would be calculating the similarity between such models. That is, the similarity degree is responsible for measuring a correspondence relation between model elements.

Many comparison techniques have been proposed in the last decades, e.g., Epsilon [5], MADMatch [6], WebDiff [7], and RCVDiff [8]. However, the state-of-the-art techniques of model comparison are designed based on rigid architecture. Which means that such techniques enable the addition of new features, but require a significant amount of effort to do so. In part, this extra effort is because of the need to restructure the entire application code to implement change requests. For this reason, developers often give up using existing approaches, creating new techniques and tools from scratch. Usually, developers spend a lot of time for implementing functions that are not related to their actual demands (e.g., defining comparison workflow)

instead of only focusing on the comparison problem at hand. Developers could perform integrations manually, but the practice of comparing and integrating models is still considered tedious, time-consuming, and error-prone. In [9], the authors highlight that a team of three analysts spent 130 man-hours to merge 25% of two variants of an end-to-end process model. In this sense, the current techniques fail on supporting model comparison in a versatile, flexible way.

Therefore, this article proposes UMLSim-Arch, a flexible architecture to support a hybrid comparison approach. The UMLSim-Arch was structured based on the design-for-change principle so that it is easy to use or extend. In this sense, an elicit set of features related to model comparison and modularized into architectural components. Developers might benefit from using UMLSim-Arch typically when performing development tasks, like computing the similarity or identifying the overlapping parts between UML class diagrams. Through the identification of highly similar design model elements, developers can invest their development efforts into grasping and reconciling the conflicting parts of design models, which could be converted into model inconsistencies, thereby improving comparison usefulness and precision.

This paper is organized as follows. Section 2 presents the related word. Section 3 shows the proposed architecture. Section 4 gives some insight about technologies that can be used to implement this architecture in practice. Finally, Section 5 presents the conclusions and future work.

2. Related Work

This section describes the related works studies, which are summarized in Table 1. To the best of our knowledge, this study is the first to explore architectural issues as a critical step to support model comparison in mainstream software projects. We have observed that both academia and industry have proposed several architectures for model comparison tools in the last decades (e.g., [5], [6], [7], [8], [10]). In [5], authors propose ECL, the Epsilon Comparison Language. This ECL enables developers to personalize comparison algorithms for adapting them to the specific metamodels. In [7], the authors propose a web tool for detecting model differences, so-called WebDiff. They propose an architecture to organize services related to model comparisons, such as parse of the input models and the computation of the similarity degree. For this, they specified a multilayer architecture to accommodate these services and user interface's components. In [8], the authors propose the RCVDiff, a tool to identify model differences. The authors are concerned with producing a common architecture for supporting the representation, visualization, and calculation of the differences of input models. In [10], the authors propose the SiDiff, framework. This framework has a kernel that enables developers to extend and adapt the metamodel, the differences algorithms, and the user's interfaces.

However, none of them has proposed modular, flexible architectures for supporting the creation of model comparison tools in practice. They do not make explicit the supported features or even the possible combinations of features that need to be established to create a valid configuration of technique.

Many studies have been proposed in the field of the model composition such as [9], [11], [12]. Specifically, in [11], authors propose a flexible, strategy-based process for model composition approaches. Their approaches are flexible because users can configure the tool to maximize results. It is strategy-based because it composes elements based on the syntactic and semantic strategies. Similarly, the authors in [12] propose a modular and flexible architecture focused on model composition. The model comparison in this work is responsible for mapping the commonalities that will be integrated into the composed output model. Therefore, the comparison step in this article is rigid and not customizable because authors implemented it in a single module.

Furthermore, many tools for model comparison were developed using multi-strategy approach to reach a more precise similarity value. It was observed that authors had implemented similar strategies such as

MADMatch [6], UMLDiff [13], and Al-Khiaty tool [14]. For example, Al-Khiaty tool and MADMatch have two comparison aspects in common, i.e., both evaluate the entity names, and neighbors of elements. In addition, both MADMatch and UMLDiff evaluate structural criteria in the similarity degree. This evidence shows that authors did not reuse the aspects present in previous approaches. Instead, they end up developing duplicated strategies from scratch. Consequently, developers could apply this effort to focus on the implementation of novel aspects instead of developing the strategies present in earlier works.

Table 1. Resume of the Related Works

Articles	Type of Architecture	Context	Year	Purpose	Architecture Details
Epsilon [5]	It does not specify.	Model comparison	2008	Authors propose a language for developers specify the properties that must be evaluated to identify the commonalities of input models	Proposes a programming Language for model comparison.
MADMatch [6]	Focus on propose a multi-strategy comparison tool	Model comparison	2013	Authors propose MADMatch, a tool to compare class diagrams using genetic algorithms.	Calculates the Similarity based on neighbors, semantic differences, and class names.
Webdiff [7]	It defines neither a modular architecture, nor variability points.	Model comparison	2011	Authors proposes an web differencing tool, called WebDiff, They proposes an architecture to organize the services related to model comparison, such as parsing the input models and the similarity calculation	Specifies a multilayer architecture. However, it Is not modular.
RCVDiff [8]	It does not specify.	Model comparison	2011	Authors propose the RCVDiff, a tool for model differencing.	The authors concerned in produce a common architecture for supporting the representation, visualization, and calculation of differences of input models
SiDiff [10]	Architecture based on interfaces,	Model comparison	2008	Authors propose the SiDiff, framework. This framework enables developers to build comparison tools. It has a kernel that enables developers to personalize the metamodel, the differentiation algorithms, and the user's interfaces.	Specifies a monolithic Kernel that provides interfaces. Developers implement these interfaces the functions.
Flexible approach [11]	Flexible process for model composition.	Model Composition	2009	Authors propose a flexible strategy-based process for model composition approaches. Their approach is flexible because users can configure the tool to maximize results. It is strategy-based because it	Proposes a model composition workflow to standardize the compositions process and guide developers to add new features

				composes elements based on the syntactic and semantic strategies.	
Farias et. al. 2015 [12]	Defines a modular and flexible architecture for model composition	Model Composition	2015	The model comparison in this work is responsible to map the commonalities to compose them in the output model. Therefore, the comparison step in this work is rigid and not customizable because is implemented in a single module.	Authors proposes a modular and flexible architecture focused on model composition
UMLDiff [13]	An algorithm to calculate structural similarities	Model Comparison	2010	Authors propose the UMLDiff, a tool that generates the models based on the source code, and compares their differences.	Calculates the Similarity based on structural, and entities names
Al-kiaty [14]	Focus on develop multi-strategy comparison tool	Model Comparison	2014	Authors develop a tool for calculate the similarity degree between UML Class Diagram	Calculates the Similarity based on neighbors, semantic differences, and class names.

3. UMLSim-Arch

This section presents the proposed architecture to support model composition of design models. For this, we describe characteristics of the UMLSim-Arch through four perspectives, including process, logical, development, and deployment one [15]. The following sections describe each aspect of this architecture. These perspectives are described as follows. Section 3.1 presents the model comparison workflow of the the UMLSim-Arch's. Section 3.2 presents the coherent vision as a feature model. Section 3.3 introduce the development perspective as a component diagram. Finally, Section 3.4 presents the deployment vision showing the architecture layers.

3.1. Intelligible Workflow for Model Comparison

Fig. 1 shows the proposed intelligible workflow for model comparison, which presents the activities performed, the artifacts generated, and the results produced. In total, the workflow has four phases that are carefully described as follows:

(1) *Analysis phase*: this step ensures the compatibility and identifies some inconsistencies of input models. The first step of this phase checks whether the types of the input models correspond. Next, it verifies whether the input models are valid [16]. The process finishes if both input models do not reach these requirements. This ensures the execution of the comparison process only if the models meet these basic requirements.

(2) *Comparison phase*: the main purpose of this step is to compare the input models in a systematic way to determine the similarity between the elements of input models [17][18]. The inputs of this step are the synonym dictionary, the comparison strategies, and the threshold. This process considers four criteria to calculate the similarity degree. These criteria specifically are: (1) Syntactic [11], the technique evaluates the structure of the visual language; (2) Semantic [6], the technique evaluates the meaning of the terms; (3) Structural [12], the approach evaluates the similarity of the aspects of the hierarchy, such as the kind of relationship and neighbors; and (4) Metrics [19][16], the technique evaluates the similarity based on quantitative attributes, such as the number of methods, and classes.

The user defines the comparison strategy choosing the aspects the calculation should consider the relevance they have in the similarity. User assigns a weight to adjust the relevance of each aspect. Two input elements are equivalent when the degree of similarity between them is equal to or greater than the threshold [11]. In addition, this step produces three outputs. The first output is a similarity matrix, indicating the degree of similarity (ranging from 0 to 1) between the elements of the input model. The second output is a description of the equivalent elements between the input models, M_A and M_B. Finally, the last output is a description of the nonequivalent elements of input models, M_A and M_B.

(3) *Visualization phase*: the main objective is to represent the equivalences according to the output data produced in the previous step. The modular aspect of this stage enables the adaptability of this comparison process to many contexts. The default strategy of this step shows a similarity matrix. This output applies in the context of model composition, where the process merges elements of the input models above the threshold. Also, developers could adjust the output to highlight the differences to identify and track inconsistent changes between input models.

(4) *Persistence phase*: this step stores the results obtained. The application can save the results in the form of states or operations. When the results are state-based, the tool saves the full diagram. The state-based techniques perform the activities based on the state. Next, the operation-based result stores the modifications made from one diagram to another, i.e., operations such as added, deleted, changed are permanently persisted. These operations apply for techniques on the versioning context to undo or transform one diagram to another.

3.2. Feature Model of UMLSim

The feature models present a general vision of the functions and characteristics of an application. It organizes a software product line, and then developers can produce various combinations of software according to their needs. Three reasons explain the adoption of this architecture. First, previous works [11], [20], [21] highlight that the need for reusable architectures to support and guide the production of new software development tools, as the technologies are constantly evolving. Second, the feature diagram represents the domain of model comparison in a modular way. Finally, the feature's diagram ensures the derivation of different products, since it contains several points of variability related to analyzing, comparing, visualizing, and persisting strategies. In this way, the proposed architecture provides the fundamental characteristics for the model comparison.

Fig. 2 shows a simplified representation of the features model of UMLSim. The points of variability of the visualization and persistence features are absent in this figure due to the space constraints. We designed the UMLSim architecture to ensure the required resources according to the comparison process described in Fig. 1. Therefore, developers must implement the required characteristics of analysis, comparison, visualization, and persistence features to generate a comparison tool. Finally, the developer must implement or adapt at least one or more comparison criteria.

3.3. Architectural Components of UMLSim

Fig. 3 shows the components that implement each feature in Fig 2. Therefore, this diagram relates the elements of Fig. 3 to the characteristics presented in Section 3.2. The small squares located on the edges of the components represent the mapping between the features to the respective elements. For example, the letter “C” at the top of the Semantic component (Fig. 3) indicates this component implements the Semantic feature (Fig. 2). Thus, the feature's design is component-based, and each feature is equivalent to a single element. The aspect-oriented programming enables the encapsulation of the features. This method allows developers are creating modularized design elements and reuse of previously implemented features.

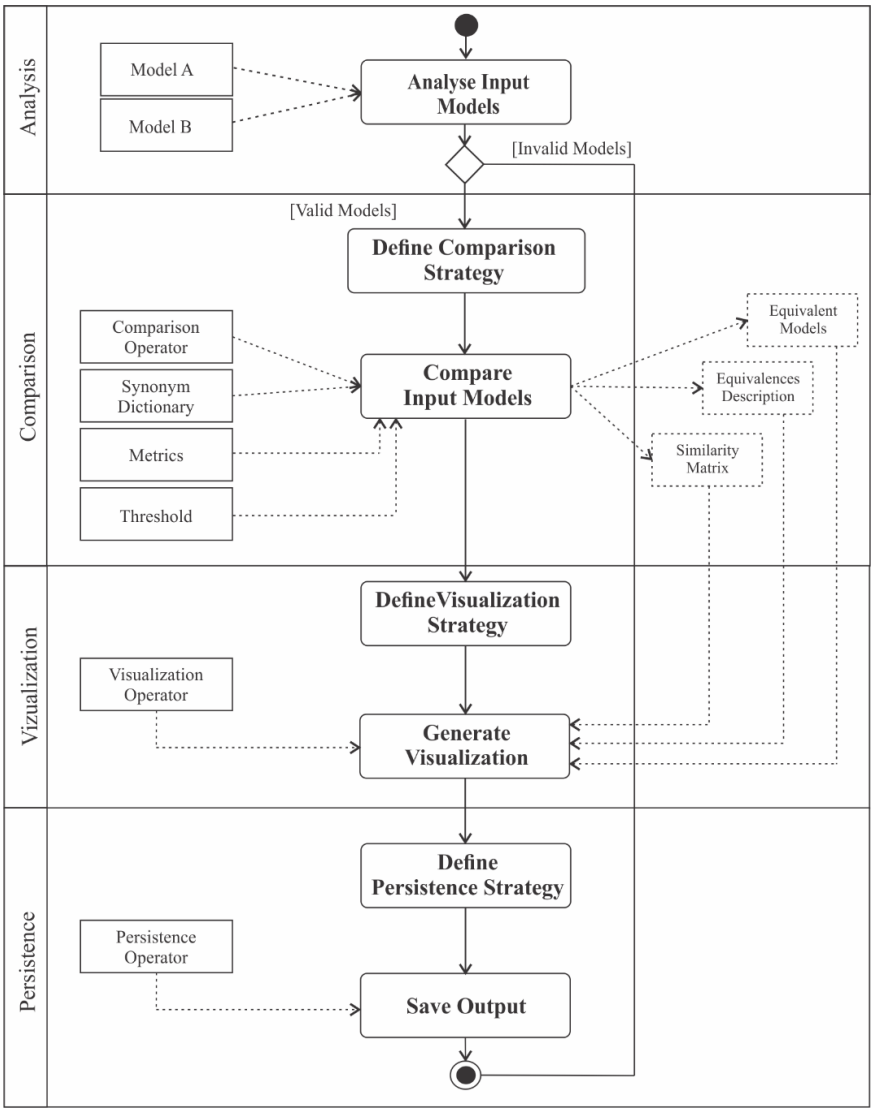

Fig. 1. Model comparison workflow.

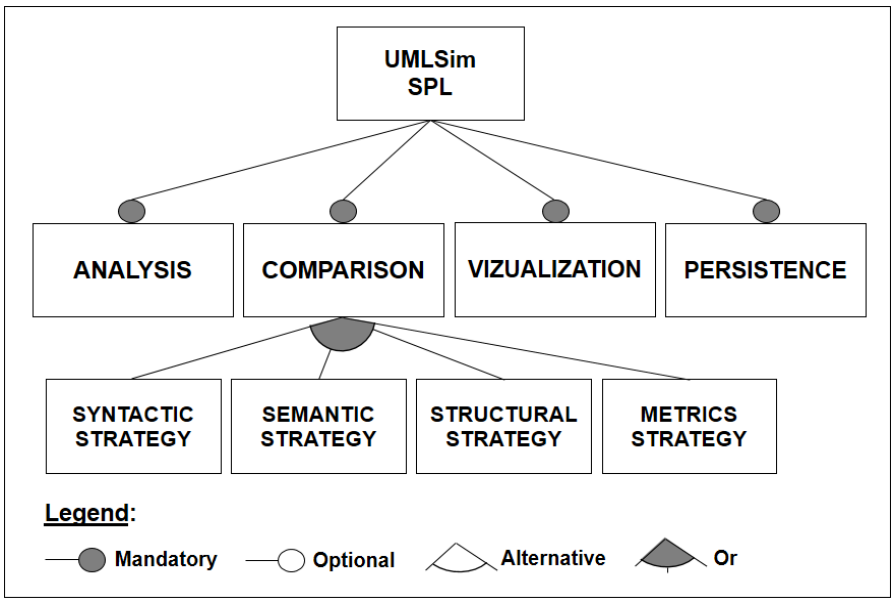

Fig. 2. Feature diagram of the UMLSim.

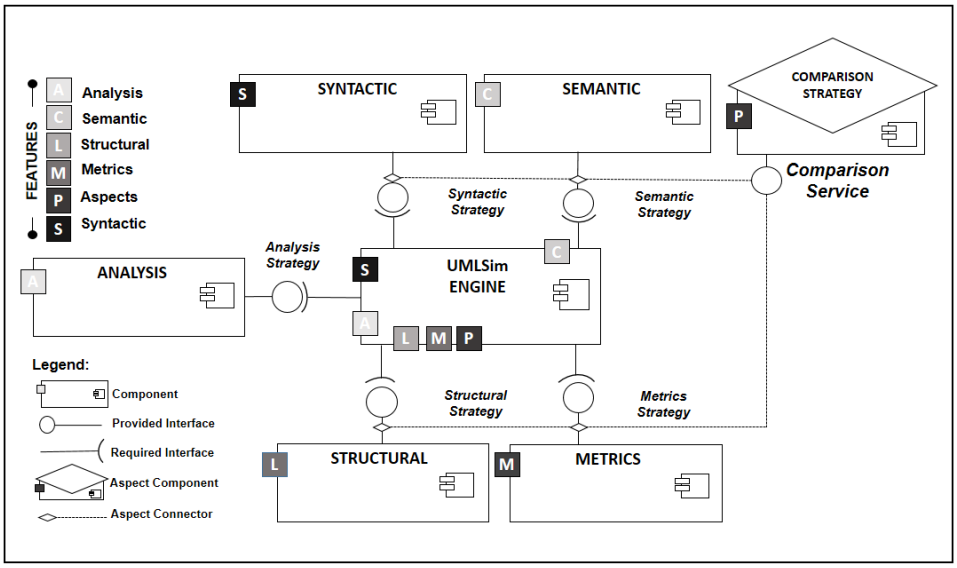

Fig. 3. Component diagram of UMlsim.

The project of the architecture conceived these components using three characteristics. First, they are standalone modules that encapsulate the behavior of elements that are responsible for implementing one (or more) resources. Second, they perform roles depending only on the interaction between their own elements. This means that the component performs the expected behavior according to their self-contained resources. Finally, they have their own provided interfaces. For example, to implement a new comparison strategy, the new component must require the comparison strategy (P) interface, and implement a new interface with UMLSim ENGINE. Another example is the addition of a new semantic comparison approach. For this, users should only require the interface of the semantic component. Furthermore, Fig. 3 focuses on the presentation of elements of a group of elements, where each element is a block performing a single role during the model comparison process.

3.4. Architectural Layers of UMLSim

The multilayer architecture of Fig. 4 shows the logical perspective of the UMLSim-Arch. Also, it also illustrates the organization of crosscutting concerns (e.g., persistence and logging) as represented by the feature's diagram. The architecture of this tool has five layers that are below described:

(1) *Presentation Layer*: it describes the application interface, receives the input data needed to perform the comparison process, then transfers those results to the application layer;

(2) *Application layer*: this layer is equivalent to the UMLSim tool's engine. It is responsible for managing the comparison process. It performs a pivotal role coordinating the requests for the operators in the process of comparison of the input models;

(3) *Variability layer*: it is responsible for implementing the variation points individually. Therefore, it is composed of the aspects that weave the individual behaviors to each element of the models (business rules layer) to the comparison operators (application layer). In practice, the aspects include in the operator's alternative or optional behaviors, which corresponds to the strategies and their rules;

(4) Business rule layer: it contains the family of algorithms that implement UMLSim comparison operators, i.e., syntactic, semantic, structural, and design metrics. These algorithms calculate the similarity degree between input models, M_A and M_B;

(5) Infrastructure Layer: it accommodates functions for handling the execution of exceptions, data access, persistence, and data logs. These features are crosscutting concerns applied during the comparison process.

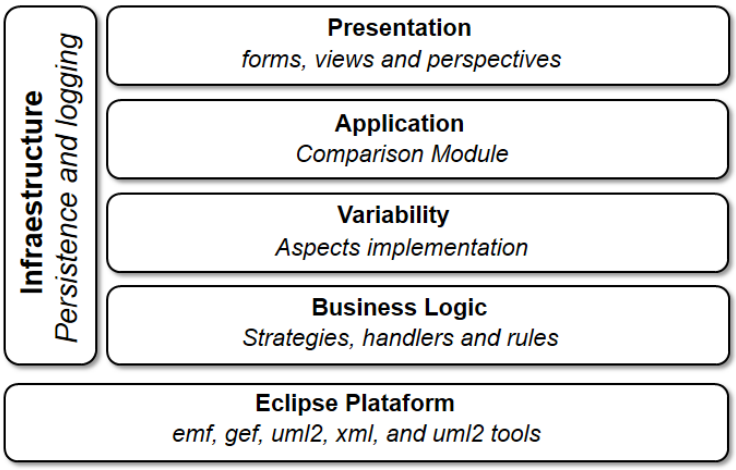

Fig. 4. Layer diagram of UMLSim.

4. Implementation Details

We have used the UMLSim-Arch to develop model comparison tools as an Eclipse plug-in. The main goal of UMLSim is to calculate the similarity between input models according to the process described in Section 3.1. The UMLSim tool was implemented using the Eclipse modeling technologies, such as EMF [22], UML2 [23], GEF [24], and UML2 tools [25] libraries.

The AspectJ [26] plug-in will be used to implement the tool's variability points, i.e., this plugin provides support to the aspect-oriented programming for ensuring the tool's flexibility. Next, the project obtains metrics from the SDMetrics API [27]. The libraries are necessary to facilitate the manipulation of diagrams, which are descriptions in XML. For example, the UML2 tool API interprets the information of the file tags described in XMI (XML for design models), transforms them into a compatible set of data, and then enabling the manipulation of the elements as objects in the Java language.

Fig. 5 presents an overview of the UMLSim prototype, where the uppercase letters from A to F represent the description of the following components: Package Explorer (A) organizes the input files in a tree structure; (B) The outline tab that shows a big-picture of the compared models; (C) the input models; (D) the console tab displaying the results from the similarity matrix; (E) The model's properties, i.e., the meta-model properties of the input models; (F) the editing palette that provides tools to the user correct and adjusts the input models before the comparison. Finally, the red and blue colors point the correspondences and the respectives similarity degree.

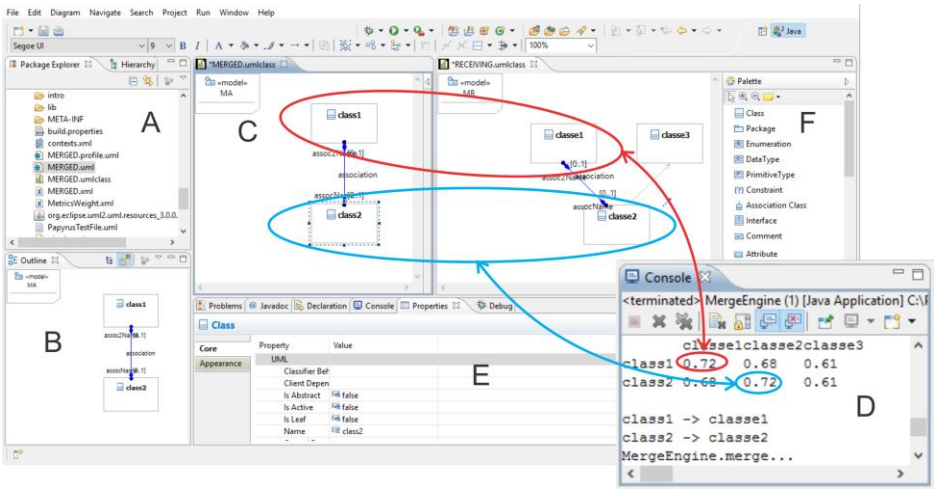

Fig. 5. Prototype of the UMLSim tool.

5. Conclusions and Future Work

This paper introduced a modular and flexible architecture for supporting the development of model comparison tools. This architecture was conceived aiming the reuse of the aspects between developers, and to provide a way to users adapt the comparison techniques according to the domain level. Moreover, a model comparison process was also developed aiming to guide developers to comprehend the essential comparison activities and their relationships more correctly. In addition, this work reported the implementation details of the UMLSim tool, a comparison tool for design models based on the UMLSim-Arch.

The next step in this research is to carefully evaluate the UMLSim tools by performing case studies to measure the precision and accuracy of the comparisons. Furthermore, the future investigations should seek to answer some questions such as: (1) do developers invest significantly more effort to develop a comparison technique from scratch than using the UMLSim? (2) How effective is this tool to compare complex UML design models? Lastly, this work represents the first step in a more ambitious agenda on better supporting the elaboration of model comparison techniques.

Acknowledgment

This work was funded by Universal project – CNPq (grant number 480468/2013-3).

References

[1] Stepha, M., & Cordy, J. R. (2016). Model-driven evaluation of software architecture quality using model clone detection. *Proceedings of the 2016 IEEE International Conference on in Software Quality, Reliability and Security* (pp. 92-99).

[2] Dong, J., Sun, Y., & Zhao, Y. (2008). Design pattern detection by template matching. *Proceedings of the 2008 ACM Symposium on Applied computing* (pp. 765-769).

[3] Adamu, A., & Zainoon, W. (2016). A framework for enhancing the retrieval of UML diagrams. *Proceedings of the International Conference on Software Reuse* (pp. 384-390). Limassol, Cyprus.

[4] Farias, K., Garcia, A., Whittle, J., Chavez, C. V. F. G., & Lucena, C. (2014). Evaluating the effort of composing design models: a controlled experiment. *Software and Systems Modeling*, *14(4)*, 1349-1365.

[5] Kolovos, D. S. (2009). Establishing correspondences between models with the epsilon comparison language. *Proceedings of the European Conference on Model Driven Architecture-Foundations and Applications* (pp. 146-157).

[6] Kpodjedo, S., Ricca, F., Galinier, P., Antoniol, G., & Gueheneuc, Y. G. (2013). Madmatch: Many-to-many approximate diagram matching for design comparison. *IEEE Transactions on Software Engineering, 39(8)*, 1090-1111.

[7] Tsantalis, N., Negara, N., & Stroulia, E. (2011). Webdiff: A generic differencing service for software artifacts. *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance* (pp. 586-589). Williamsburg, VA, USA.

[8] Van, D. B. M., Protic, Z., & Verhoeff, T. (2010). RCVDiff-a stand-alone tool for representation, calculation and visualization of model differences. *Proceedings of the international workshop on models and evolution-ME co-located with ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems*. Oslo, Norway.

[9] La Rosa, M., Dumas, M., Uba, R., & Dijkman, R. (2013). Business process model merging: An approach to business process consolidation. *ACM Transactions on Software Engineering and Methodology* (TOSEM), *22(2)*.

[10] Schmidt, M., & Gloetzner, T. (2008). Constructing difference tools for models using the SiDiff framework.

Proceedings of the Companion of the 30th International Conference on Software Engineering (pp. 947-948).

[11] Oliveira, K. F., Breitman, K. K., & Oliveira, T. O. (2009). A flexible strategy-based model comparison approach: Bridging the syntactic and semantic gap. *Journal of Universal Computer Science, 15(11),* 2225-2253.

[12] Farias, K., Gonçales, L., Scholl, M., Oliveira, T. C., & Veronez, M. (2015). Toward an architecture for model composition techniques. *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering* (pp. 656-659).

[13] Xing, Z. (2010). Model comparison with GenericDiff. *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering* (pp. 135-138).

[14] Al-Khiaty, M. A. R., & Ahmed, M. (2016). UML class diagrams: Similarity aspects and matching. *Lecture Notes on Software Engineering*, *4(1)*, pp. 41-47.

[15] Kruchten, P. B. (1995). The 4+ 1 view model of architecture. *IEEE Software*, *12(6)*, 42-50.

[16] Mishra, A. K., & Yadav, D. K. (2015). Validation of UML design model. *Journal of Software*, *10(12)*, 1359-1366.

[17] Ohst, D., Welle, M., & Kelter, U. (2003). Differences between versions of UML diagrams. *ACM SIGSOFT Software Engineering Notes*. *28(5)*.

[18] Kelter, U., Wehren, J., & Niere, J. (2005). A generic difference algorithm for UML models. *Software Engineering, 64(105-116)*. pp 4-9.

[19] AbuHassan, A., & Alshayeb, M. (2016). A metrics suite for UML model stability. *Software & Systems Modeling*, pp. 1-27.

[20] Clarke, S. (2001). Composition of object-oriented software design models (Doctoral dissertation, Dublin City University).

[21] Farias K. (2012). Empirical evaluation of effort on composing design models (doctoral DISSERTATION, PUC-rio).

[22] EMF. Eclipse modeling framework. (2017), Retrieved on March 15, 2017, from https://eclipse.org/modeling/emf/

[23] UML2. (2017). Retrieved on March 15, 2017, from http://www.eclipse.org/modeling/mdt/downloads/?project=

[24] GMF. graphical modeling framework**.** (2017), Retrieved on March 15, 2017, from https://eclipse.org/modeling/ emf/

[25] UML2 TOOLS. (2017). Retrieved on March 15, 2017, from http://www.eclipse.org/modeling/mdt/downloads/?projec t=uml2tools

[26] ASPECTJ. AspectJ. (2017). Retrieved on March 15, 2017, from https://eclipse.org/aspectj/

[27] SDMETRICS. SDMetrics the Software Design Metrics tool for the UML. (2017). Retrieved on March 15, 2017, from http://www.sdmetrics.com/index.html

Lucian Gonçales is a master student in the interdisciplinary graduate program on applied computing (PIPCA) at the University of Vale dos Rio dos Sinos (Unisinos). He completed his undergraduate studies in Computer Science at the University of Vale do Rio dos Sinos (UNISINOS) in 2013. He also received his formation as a computing technician from Sao Lucas Education Institute in 2006. His current research is about software modeling, empirical evaluation of model composition techniques, and software metrics. His current research interests include model matching, software metrics, and neuroscience applied to software engineering.

Kleinner Farias is an assistant professor in the interdisciplinary postgraduate program in applied computing at the University of Vale dos Rio dos Sinos (Unisinos). He is an associate member of the OPUS Researcher Group at the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil. He received his Ph.D. in computer science from PUC-Rio in 2012. He received his master's degree in computer science from the Pontifical Catholic University of Rio Grande do Sul (PUC-RS) in 2008. He completed his undergraduate studies in Computer Science at the Federal University of Alagoas and in Information Technology at the Federal Institute of Alagoas in 2006. His current research interests include software modeling, empirical evaluation of model composition techniques, model-driven software development, software metrics and software product lines.

Vinicius Bischoff is a master student at University of Vale do Rio dos Sinos (UNISINOS) since 2015. He received his Bachelor degree in Information Systems in 2012 at Integrated Taquara's Faculties (FACCAT). He obtained his specialization in engineering of software testing in the Federal University of Pernambuco (UFPE) in 2013. Thus, He worked as a test engineer at Motorola Mobility between 2013 and 2014. He also received his Bachelor degree on Transportation Engineering in 2005 in the Catholic Pontifical University of Rio Grande do Sul (PUC-RS). Since 2015, he started are researching the subject of integration of feature models in the interdisciplinary graduate program in applied computing (PIPCA). His current research interests include controlled experiments in the software engineering, and efficacy techniques for software testing.

Matheus Segalotto is a master Student in the interdisciplinary graduate program on applied computing (PIPCA) at the University of Vale dos Rio dos Sinos (Unisinos). He completed his undergraduate studies in analysis and systems development at the University of Vale do Rio dos Sinos (UNISINOS) in 2014. He also received his formation as a technician systems development from Fundação Liberato Salzano Vieira da Cunha in 2010. His current research is about software comprehension, data mining, and neurophysiological indicators. His current research interests include machine learning, big data, and neuroscience applied to software engineering.