International Journal of Computer Science and Software Engineering (IJCSSE), Volume 5, Issue 12, December 2016
ISSN (Online): 2409-4285 www.IJCSSE.org Page: 260-264

Detecting Inconsistencies in Multi-view UML Models

Vanessa Weber[1], Kleinner Farias[2], Lucian Gonçales[3], Vinícius Bischoff[4]

[1, 2, 3, 4] Interdisciplinary Graduate Program in Applied Computing (PIPCA), University of Vale do Rio dos Sinos (UNISINOS), São Leopoldo, Rio Grande do Sul, Brazil

[1]weber.nessa@gmail.com, [2]kleinnerfarias@unisinos.br, [3]lucianjosegoncales@gmail.com, [4]viniciusbischoff@gmail.com

ABSTRACT

Inconsistencies in conflicting multi-view UML models can be major obstacles to the quality and productivity of software development. In the current literature it can be observed that some tools were developed to support the detection of inconsistencies, but none of them are still consolidated. In addition, many of these tools only evaluate syntactic inconsistencies, not considering semantic ones. The tools available are often unable to detect syntactic and semantic inconsistencies in conflicting multi-view UML models. To address this issue, we propose DIUML, a tool that includes: (i) detection of inconsistencies in multi-view UML models through design metrics; (ii) detection of syntactic and semantic inconsistencies, indicating objects and classes affected by them; and (iii) evaluation of the severities of each type of inconsistency detected. Our preliminary evaluation indicated that DIUML was able to detect inconsistencies in multi-view UML models with 337 elements from 10 different combinations of UML class and sequence diagrams.

Keywords: *Inconsistencies Detection, Multi-view UML Models, Tool.*

1. INTRODUCTION

Unified Modeling Language (UML) is the de facto standard for object-oriented software modeling [3, 4, 12] and has been widely used to represent design projects through a multi-view approach. According to [12], the UML seeks to advance the state of the industry by allowing the interoperability of the visual object-oriented modeling tool. To do this, the UML specification provides a set of human-readable notation elements, as well as rules for combining them into various types of diagrams, considering structural and behavioral aspects of the software system under development.

In collaborative software development, for example, virtual teams can simultaneously work on partial views of a general architecture by editing structural aspects, e.g., manipulating UML class diagrams [15, 16, 17], while team members can edit behavioral aspects, e.g., modifying UML sequence. However, at some point they need to consider both aspects to fully understand design decisions for creating an overview of the overall architecture. To do this, developers must obtain information from UML class and sequence diagrams. Unfortunately, the diagrams end up suffering from conflict problems due to contradicting modifications realized in parallel in overlapping model elements. This means that modifications made in the UML class diagram can generate a mismatch with model elements in the UML sequence diagram. Therefore, developers have to lead with design models with inconsistencies. Empirical studies (e.g., [3, 9]) report that inconsistencies in multi-view UML design models can harm the correct understanding of design decision, leading to misinterpretation of them. In [1], the authors report that the lower quality of UML models is correlated with lower quality of the source code.

To overcome these issues, this paper presents the DIUML, a tool that detects inconsistencies in multi-view UML models. Developers and system analysts can benefit from using DIUML when performing software maintenance tasks for implementing change requests, or even developing new features. Detecting inconsistencies in multi-view UML models, developers will be able to seek solutions to the inconsistency problem, rather than taking into account improper representations of design decisions, which can lead to misinterpretations.

To detect inconsistencies in conflicting multi-view UML models, the novelty of DIUML is the detection of inconsistencies in multi-view UML models through design metrics [18, 19], the detection of syntactic and semantic inconsistencies, indicating objects and classes affected by them, and the evaluation of the severities of each type of inconsistency detected. The DIUML is implemented in C # through Visual Studio and runs as an executable. Preliminary results of the use of DIUML indicated that it was able to detect inconsistencies in UML models with 337 elements of 10 different UML class and sequence Diagrams. These results indicate that our tool can improve the quality and productivity of software development by mitigating the misinterpretation problem of design decisions caused by undetected inconsistencies.

The remainder of the paper is organized as follows. Section 2 outlines the main concepts that are going to be used and discussed throughout the paper. Section 3 briefly compares this work with others, presenting some differences and commonalities. Section 4 presents the proposed DIUML tool. Section 5 discusses the design and implementation details. Section 6 presents a scenario of use of the proposed tool. Section 7 presents some concluding remarks and future work.

2. BACKGROUND

We have identified two broad categories of inconsistencies, such as: (1) *syntactical inconsistencies* that arise when project models are not in accordance with the metamodel language; and (2) *semantic inconsistencies* where the meaning of the model element does not correspond to the actual design model. Current literature has identified a number of inconsistencies (e.g., [8, 9]), which have also been used in previous empirical studies reported in [9]:

- Abstract classes in the sequence diagram (CaSD): the UML metamodeling [12] makes it clear that abstract classes should not be represented in the sequence diagram, since single instances of concrete classes are represented in the following diagram carefully to represent the method call exchanged between them. This is due to abstract classes cannot be instantiated, so objects derived from abstract classes can not be produced [8].
- Unnamed message (EnN): each message sequence diagram must have a specific name; otherwise, it is impossible to define which methods should be called in the class diagram [6, 8].
- Message without method (EcM): objects in the sequence diagram of message exchange to execute scenarios. These messages are actually calling method, which must be defined in the class diagram [6].
- Multiple class definitions with the same name (Cm): When a class is instantiated more than once with the same name in the same or different diagrams in a single UML model.
- Definitions of multiple objects with the same name (Om): for inconsistency of this type, we can see more than one object with the same name in the same sequence diagram.
- Class not instantiated in SD (CnSD): when there is an object instantiated in the UML sequence diagram for all the classes present in the class diagram, it can be said that there is an inconsistency of this type.
- Object not instantiated in CD (CnCD): when for an object instantiated in the sequence diagram there is no corresponding class in the UML class diagram, it results in this type of inconsistency.
- Message in the wrong direction (ED): this type of inconsistency occurs when an object in an UML sequence diagram calls a method of a wrong object. This is a case where the message name does not match its method.

3. RELATED WORKS

Researchers and practitioners have widely recognized that metrics can be used as indicator to identify inconsistencies and measure the degree of incompleteness of UML models [8]. Thus, design metrics can help to measure design models by quantifying key features, mainly ones previously defined in UML metamodel. However, design metrics have not been used for supporting the detection of inconsistencies of UML models. Instead, authors have focused on using syntactic [13], and semantic [5] facets to verify inconsistencies on design models only.

Moreover, there is a lack of automated methods for checking the inconsistencies of multi-view UML models, before developers use them in production environment. In [11] authors propose a tool that takes into account multi-view models for verifying inconsistencies. But, in practice, the current literature is still limited to techniques that check inconsistencies in UML class diagram [14], or in UML sequence diagram [10], not both.

In [20, 21], the authors present a literature review about model comparison, highlighting the main techniques in the current literature. In [22], the authors come up with, in turn, a technique to compare UML model aided by ontologies.

To sum up, little has been done to detect the inconsistencies using well-established design metrics. Therefore, to the best of our knowledge, this work is the first to (1) consider metrics for detecting the inconsistencies of UML models, and (2) provide a tool that execute the inconsistency detection automatically.

4. DIUML TOOL

The process of detecting inconsistencies in UML models of DIUML has two premises. The first includes the detection of inconsistencies in multi-view UML models through metrics and algorithms using mathematical logic. The second classifies each type of inconsistencies

detected based on the level of severity (i.e., low, medium and high), thus supporting decision making by developers and system analysts.

The detection of inconsistencies in multi-view UML models uses measures of well-established design metrics. These measures are computed using the SDMetrics[1], a software design metrics tool for UML models. The measures are stored in XML files. For a syntactic and semantic validation of the combination of UML class and sequence diagrams, the DIUML tool stores the names of the elements (class or object), the methods and messages, the relationships between the elements, and the classes have the property *isAbstract* equal to *true*. Code 1 shows how this procedure is computed.

Code. 1. Reading logic of XML file variables

```
1. Var class = new Class();
2. Var object = new object();
3. Class.Name = ElementClasse.Attribute("name"). Value;
4. Class.Abstrata = bool.Parse(elementClasse.Attribute(
   "isAbstract").Value);
5. Class.Id = ClassClasse.Attribute("xmi.id").Value;
6. Var relationships = elementClasse.Descendants(ns +
   "ModelElement.clientDependency").FirstOrDefault();
```

The DIUML tool detects the two-macro types of inconsistencies in the multi-view models, i.e., syntactic and semantic inconsistencies. For the detection of syntactic inconsistencies, simple metrics such as counters and paired combinations were used. For the detection of semantic inconsistencies, it was required to use the values of properties, such as names, methods, messages and IDs. Once the XML files of the class and sequence diagrams have been read, the tool applies, in a linear fashion, each of the validations of algorithms for the different types of inconsistencies. Code 2 shows how this validation is performed.

Code. 2. Example of procedure to detect inconsistency.

```
1. Public void InconsistencyTypeCnCD() {
2.      Var inconsistent = Sequence.Classes.Where(c =>! Class.
        Classes.Any(s => s.Name == c.Name));
3.      Foreach (var inconsistent in inconsistent) {
            If (! Inconsistent.Insistencies.Any (i => i ==
            InconsistencyType .CnCD)) {
4.              Inconsistent.Insistencies.Add(Inconsistency Type.
                CnCD); }}}
```

Code 2 identifies each object in the UML sequence diagram and verifies if there is a class in the corresponding UML class diagram. If the result is different from 1 to 1, this type of inconsistency will be present. Since the types of inconsistencies in UML class and sequence diagrams have been detected, the DIUML tool presents the results, including an analysis of the severity of each type of inconsistency. Code 3 shows how the severity of different types of inconsistencies was computed.

[1] SDMetrics: http://www.sdmetrics.com/

Code. 3. Procedure defined to detect inconsistency.

```
1.  Case Type.Inconsistency.CnSD:
    Case Type.Inconsistency.EnN:
    Case Type.Inconsistency.CnCD:
    {Return "High";}
2.  Case Type.Inconsistency.EcM:
    Case Type.Inconsistency.ED:
    Case Type.Inconsistency.CaSD:
    Return "Medium";}
3.  Case Type.Inconsistency.Cm:
    Case Type.Inconsistency.Om:
    {Return "Low";}
```

Thus, the tool implements the reading, validation and interpretation of the XML files and applies the eight types of algorithms to detect inconsistencies, thus generating the final result of the detection of inconsistencies and classifying the severities, in order to allow the interpretation of the data. Having the severity of the inconsistencies at hand, developers can prioritize, or even overlook the inconsistency detected.

5. DESIGN AND IMPLEMENTATION DETAILS

Figure 1 shows a representation of the DIUML design, and how its main elements are related. The tool has two inputs, i.e., UML class and sequence diagrams, both are XML files. These XML files can be generated from UML modeling tools by exporting the diagrams used in XML.

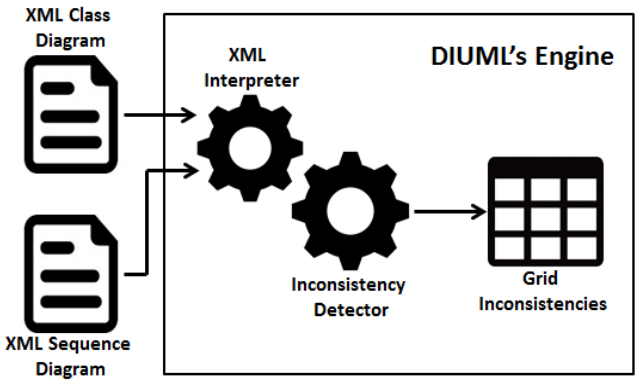


Fig. 1. DIUML Design.

The above entries are processed by DIUML's engine to detect inconsistencies in the multi-view UML models, comparing the data between the class and sequence diagrams provided. The DIUM was developed with the Microsoft Visual Studio platform using the C# language. It reads the XML files and stores in memory the data collected. The tool can be run in Windows or Linux platforms. After XML files have been read, the DIUML tool analyzes each type of inconsistency (listed in

Section 2). The next step is to organize the detected inconsistencies in a grid. Code 4 presents the procedure defined to generate a grid with the list of inconsistencies detected.

Code. 4. Procedure to display the detected inconsistency in a grid.

```
1. Type = diagramType == diagramType.Class? “Class”: “Object”
2. Name = class.Name
3. Inconsistency = inconsistencyHelper.ObterInconsistency
   (inconsistency? “NameInconsistencies”)
4. Severity = inconsistencyHelper.obterGravity (inconsistency?
   “High”: “Medium”: “Low”)
```

6. SCENARIO OF USE

To run DIUML, we need to provide the UML class and sequence diagrams in XML format, as described in section 5. Input files do not need to be in a specific location on the machine because the tool allows the search the files using the explorer standard. The required entries are the Class and Sequence Diagrams in XML format, which can be generated by the UML modeling tool itself, such as Astah Pro [3] or Enterprise Architect [11].

The tool does not need to be installed on the user’s computer and does not require any database configuration, since it is an executable file. This makes it easy to use on different computers and environments, such as Windows and Linux operating systems. The DIUML does not require any settings or parameter changes. The user just locates and run the executable file, namely DIUML.exe.

To illustrate the use of the tool, the XML files of the Class and Sequence Diagrams presented in Figure 2 and Figure 3 will be read and analyzed:

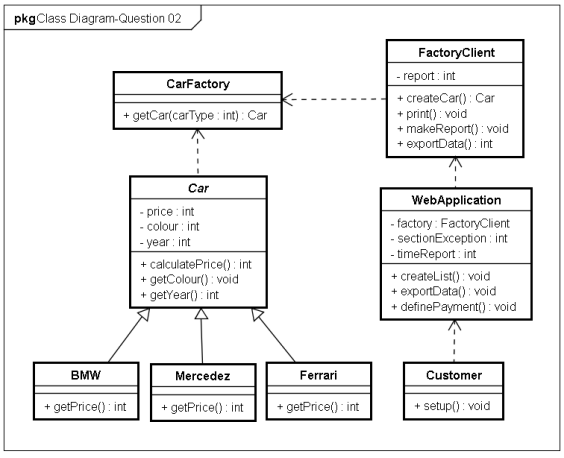


Fig. 2. DIUML Class Diagram.

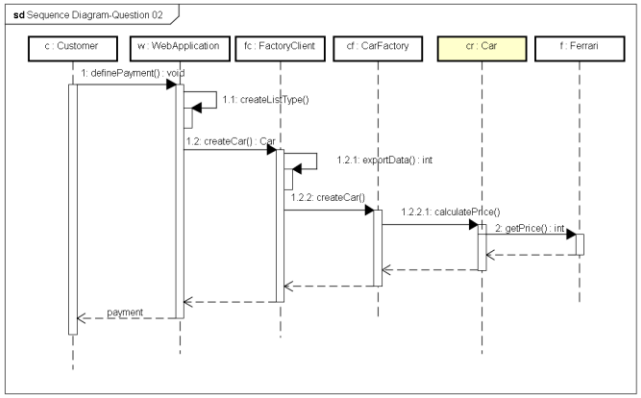


Fig. 3. DIUML Sequence Diagram.

After the tool has been started, the DIUML shows the screen shown in Figure 4, in which the user must inform the two diagrams that will be evaluated.

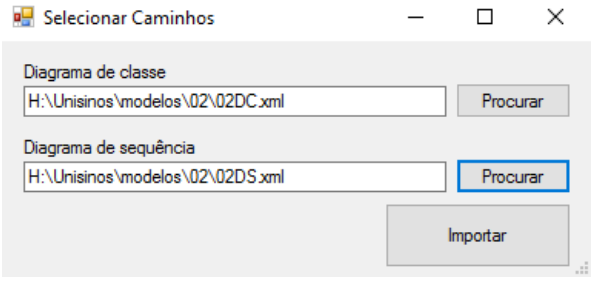
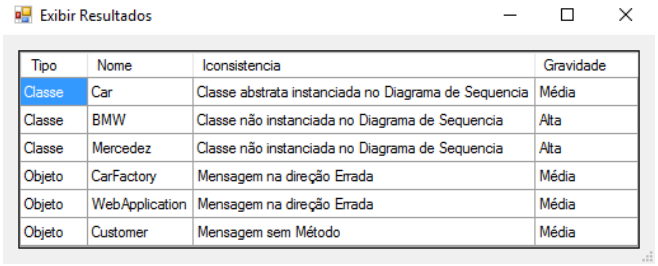


Fig. 4. DIUML Files Selector.

When the user selects the “Importar” option, the tool automatically reads the XML files and stores in the local memory. Once the reading and detection of the inconsistencies is done, the tool presents the screen shown in Figure 5, i.e., a grid with the elements in which inconsistencies were found and which inconsistencies were detected, as well as the severity of each type of inconsistency.



Tipo	Nome	Iconsistencia	Gravidade
Classe	Car	Classe abstrata instanciada no Diagrama de Sequencia	Média
Classe	BMW	Classe não instanciada no Diagrama de Sequencia	Alta
Classe	Mercedez	Classe não instanciada no Diagrama de Sequencia	Alta
Objeto	CarFactory	Mensagem na direção Errada	Média
Objeto	WebApplication	Mensagem na direção Errada	Média
Objeto	Customer	Mensagem sem Método	Média

Fig. 5. DIUML Output.

7. CONCLUSIONS AND FUTURE WORK

This article presented the DIUML, a tool to detect inconsistencies in multi-view UML models. Our initial results indicated that DIUML was able to properly detect the inconsistencies explained in Section 2, providing concrete evidence of tool usefulness. As a future work, we seek to adjust the tool so that the code of detection of inconsistencies can become incremental and allow the configuration of new types of inconsistencies.

8. ACKNOWLEDGMENTS

This work was funded by CNPq Universal Project 14/2013 (grant number 480468/2013-3), Brazil.

REFERENCES

[1] A. Nugroho, B. Flaton, M. Chaudron, "Empirical analysis of the relation between level of detail in UML models and defect density". In: 11th international conference on Model Driven Engineering Languages and Systems. Springer, 2008, pp. 600-614.

[2] Astah Pro. Available on <http://www.astah.net/editions/professional>.

[3] M. Chaudron, W. Heijstek, A. Nugroho, "How effective is UML modeling?". Software and Systems Modeling 2012, Vol. 11, n. 4, pp. 571-580.

[4] W. J. Dzidek, E. Arisholm, L. C. Briand, 2008. "A realistic empirical evaluation of the costs and benefits of UML in software maintenance". IEEE Transactions on Software Engineering, 2008, vol. 34, n. 3, pp. 407-432.

[5] A. Egyed, "Automatically detecting and tracking inconsistencies in software design models". IEEE Transactions on Software Engineering. 2011. Vol. 37 n. 2, pp. 188-204.

[6] K. Farias, A. Garcia, J. Whittle, C. Chavez, C. Lucena, "Evaluating the effort of composing design models: a controlled experiment". Software & Systems Modeling, 2015, vol. 14, n. 4, pp. 1349-1365.

[7] Enterprise Architect. <http://www.sparxsystems.com/products/ea/>

[8] C. Lange, "Assessing and improving the quality of modeling a series of empirical studies about the UML" Ph.D. thesis, Technische Universiteit Eindhoven, Eindhoven. 2007.

[9] C. Lange, M. Chaudron, "Effects of defects in UML models: an experimental investigation." In: 28th international Conference on Software Engineering. 2006. Shanghai, China, pp. 401-411.

[10] X. Li, J. Lilius, T. Centre, C. Science, "Checking compositions of UML sequence diagrams for timing inconsistency". In: 7th Asia-Pacific Software Engineering Conference (APSEC). 2000. pp. 154-161.

[11] R. E. Lopez-Herrejon, A. Egyed, "Detecting inconsistencies in multi-view models with variability." In: Kühne, T., Selic, B., Gervais, M.-P., Terrier, F. (Eds.), European Conference on Modelling Foundations and Applications. 2010. Paris, France, pp. 217-232.

[12] OMG, 2015. UML: Infrastructure specification, version 2.5. URL http://www.omg.org/spec/UML/2.5/PDF

[13] A. Reder, A. Egyed, "Computing repair trees for resolving inconsistencies in design models." In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. ASE 2012. ACM, New York, NY, USA, pp. 220-229.

[14] S. S. Satish, S. R. Shashikant, V. K. Sambhe, R. B. Shelke, G. Kocharekar, "A minimum cardinality consistency-checking algorithm for UML class diagrams." In: International Conference and Workshop on Emerging Trends in Technology. ICWET '10. 2010 pp. 222-223.

[15] K. Farias, A. Garcia, J. Whittle, C. Lucena, "Analyzing the Effort of Composing Design Models of Large-Scale Software in Industrial Case Studies". 16th International Conference on Model-Driven Engineering Languages and Systems (MODELS'13), pp. 639-655, Miami, USA, September 2013.

[16] E. Guimarães, A. Garcia, K. Farias, "On the Impact of Obliviousness and Quantification on Model Composition Effort". 29th Symposium On Applied Computing (SAC.14), Gyeongju, Korea, March, 2014.

[17] K. Farias, A. Garcia, J. Whittle, C. Chavez, C. Lucena, "Evaluating the Effort of Composing Design Models: A Controlled Experiment". 15th International Conference on Model-Driven Engineering Languages and Systems (MODELS'12), Vol. 7590, pages 676-691, Innsbruck, Austria, 2012.

[18] K. Farias, A. Garcia, J. Whittle, "On the Quantitative Assessment of Class Model Compositions: An Exploratory Study". In: Empirical Studies of Model-Driven Engineering (ESMDE'08) co-located MODELS'08, Vol. 1, pp. 1-10, Toulouse, France, 2008.

[19] K. Farias, A. Garcia, C. Lucena, "Effects of Stability on Model Composition Effort: an Exploratory Study". Journal on Software and Systems Modeling, Vol. 13, No. 4, pages 1473–1494, 2014.

[20] L. Gonçales, K. Farias, M. Scholl, T. C. Oliveira, M. Veronez, "Model Comparison: a Systematic Mapping Study". 27th International Conference on Software Engineering and Knowledge Engineering, pages 546-551, Pittsburgh, PA, USA, July, 2015.

[21] L. Gonçales, K. Farias, M. Scholl, M. Veronez, T. C. Oliveira, "Comparison of Design Models: A Systematic Mapping Study". International Journal of Software Engineering and Knowledge Engineering, Vol. 25, pages 1765-1770, 2015.

[22] K. Oliveira, K. Breitman, T. Oliveira, "Ontology Aided Model Comparison". 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'09), pp. 78-83, Potsdam, Germany, 2009.