Extensible Real Time Software Design Inconsistency Checker: A Model Driven Approach

[1]G. Ramesh [2]T. V. Rajini Kanth [3]A. Ananda Rao, *Member, IAENG*

***Abstract*—Unidentified inconsistencies in software design models can obstruct the desired time-to-market attribute of software. The consequences of them are far reaching besides being reflected in the implementation. Detecting and tracking inconsistencies in model-driven software development has been challenging. However, there is an encouraging fact that the emergence of Unified Modelling Language (UML) brought uniform notations to be used across the globe. Many UML based design consistency checkers came into existence. However an effective automated approach for consistency checking is still desired as it can lead to early detection of inconsistencies. Thus frequently occurring cost and budget overruns in software industry can be eliminated. Towards this end this paper implemented a framework named Extensible Real Time Software Design Inconsistency Checker (XRTSDIC). The framework enables software engineers to have real time feedback on model inconsistencies throw light into the issues early as it is wise to use a stitch in time and avoid eight. The consistency checker not only exploits consistency rules but also viewpoints besides supporting a degree of formal tolerance for inconsistencies. This makes the proposed system flexible with user description while resolving inconsistencies. Our empirical evaluation shows that implemented approach is comparable with previous approaches with significant improvement in speed, scalability and accuracy in detecting inconsistencies in software design models. The prototype demonstrates the proof of concept.**

Index Terms – tracking inconsistencies, unified modelling language, software design models.

I. INTRODUCTION

Model Driven Engineering (MDE) is a widely used approach in software development. Most of the software development companies across the globe use UML for object oriented analysis and design. Based on UML specifications provided by Object Management Group (OMG), many vendors are providing UML modelling tools that can be used by software engineers to design or model any software before implementing it. In fact software engineers might use diversified modelling tools based on their acquaintance on particular tool or other requirement. As the software development is based on the modelling of software systems, it is essential to validate models before transforming them into the next phase in software development life cycle. Stated differently, the inconsistencies in the design model will eventually reflect in the software implementation or coding. The inconsistencies identified in the coding phase prove costly in terms of time and effort for fixing them. This result in budget overruns increased cost and thus may cause failure as well. Therefore it is essential to detect and track inconsistencies in design itself and make necessary changes early in the life cycle.

Many approaches came into existence for consistency checking in software design models. We studied many models found in the literature. From the review of literature we found that all solutions are focusing on the method followed to identify inconsistencies. We believed that there is need for building a comprehensive framework that can cater to the diversified needs of software engineers with flexible personalized configuration for choosing modelling tool, language for consistency rules and visualization methods. Towards this end we proposed and implemented a framework that is scalable, flexible, and supports real time detection and tracking of model inconsistencies.

Our contributions in the paper include that we proposed a framework named Extensible Real Time Software Design Inconsistency Checker (XRTSDIC) which has provision for personalized configuration and execution model. This framework is flexible, scalable and supports extensible features in terms of supporting modelling tools, consistency rules and visualization techniques. To our knowledge it is for the first time we proposed such comprehensive framework using which model construction and consistency checking can be done using the modelling tool of user's choice. As the framework is extensible, it becomes more and more flexible and useful as new possibilities are added to it. The remainder of the paper is structured as follows. Section II reviews literature on prior works on consistency checking. Section III provides preliminaries. Section IV presents the proposed framework. Section V provides prototype implementation. Section VI provides experimental results while section VII concludes the paper besides providing directions for future work.

II. RELATED WORKS

Many researchers contributed towards consistency checking in software design models. Nentwich, Emmerich, and Finkelstein [6] explored static consistency checking using first-order-logic to represent relationships between elements of XML documents that correspond to UML models. Mens et al. [7] proposed an approach that analyzes dependencies among various resolution rules. They employed transformation dependency analysis to detect and resolve inconsistencies in design models. Heckel et al. [8] explored view-oriented approach in identifying inconsistencies in design models. They tried to translate UML models into semantic domains in order to check inconsistencies.

Manuscript submitted on December 8, 2015.
[1] Mr. G. Ramesh is Research Scholar with JNTUA, Ananthapuramu, Andhra Pradesh, India (phone: 0440862112; e-mail:ramesh680@gmail.com).
[2] Dr. T.. V. Rajini Kanth is professor in CSE with SNIST Ghatkesar, Hyderabad, Telangana State, India (e-mail: rajinitv@gmail.com).
[3]Dr. A. Ananda Rao is Director and professor in CSE with JNTUA, Ananthapuramu, Andhra Pradesh, India (e-mail: akepogu@gmail.com).

Kaneiwa and Satoh [9] focused on detecting inconsistencies in UML class diagrams. They achieved it by translating the classes into first-order-predicate logic. Towards this end they defined algorithms that can check inconsistencies among restricted UML class diagrams. Kielland [10] employed XML Metadata Interchange (XMI) format for checking inconsistencies of UML models. Hnatkowska et al. [11] employed Object Constraint Language (OCL) for formulating consistency conditions that can be used to find discrepancies between components of UML model. Baclawski et al. [12] proposed a method that checks consistencies in ontologies modelled using UML. Pap et al. [13] explored ways and means to find inconsistencies in state chart specifications. Their focus was on consistency based on structure which is static and safety related reachability attributes verified dynamically. Khan and Porres [14] logic reasoners built using web ontology language named OWL for inconsistency checking in UML models. Though the scope of their approach is limited it is fully automatic solution. Straeten *et al*. [15] employed description logic to ascertain inconsistencies among UML models and concluded that description logic tools can play significant role in checking inconsistencies in software design models. Similar kind of research was carried out in [16] and [28].

Zhao *et al*. [17] proposed a formalism known as Split Automata that is employed to check consistency between sequence and state chart diagrams. Diskin *et al*. [18] conceived software design model as a collection of views or models that contain local elements. As these models overlap they might have inconsistencies when a global constraint set is considered. They proposed an approach that explores global consistency checking. Egyed [19] presented an automatic approach that helps in determining consistency rules to be applied automatically when model elements change. Their approach gives appropriate feedback as model elements are being drawn. Gryce *et al*. [20] employed xlinkit tool for checking consistency of UML models. They proposed an approach that makes use of xlinkit to achieve this. Graph transformation approach was used by Chama *et al*. [22] for model checking. They focused on both static and dynamic models for inconsistency checking. It was a meta-modelling approach that could help in model checking. Blanc *et al*. [23] proposed an operation-based model construction that could help in detecting inconsistencies in design models.

In the literature, Petri Nets is found to be an alternative approach for model checking. Thierry-Mieg *et al*. [24] employed instantiable Petri Nets to capture model dynamics accurately before checking inconsistencies. Sourrouille and Caplat [25] explored a pragmatic approach to checking UML models for inconsistencies. They concluded that transformation of UML models into some kind of formal language can help in consistency checking. Defining consistency rules play a vital role in model checking as UML tools do not provide them by default. A formal method for specifying constancy rules was explored in [26]. Similar kind of work can be found in [1] as well. Egyed [27] built a tool for model checking. The tool was named View Integra which proved to be scalable. Egyed [29] proposed an analyzer tool for instance checking of inconsistencies in UML models. The tools reviewed in this section were useful in inconsistency checking. However, we felt a comprehensive and flexible tool that can provide personalized configuration and execution model as significant underlying parts is desired. In this paper we proposed such framework.

III. PRELIMINARIES

This section provides details on preliminary information that can help understand the proposed approach. The subsections provide details of inconsistencies, various consistency rules pertaining to UML diagrams such as use case, sequence, collaboration and state chart besides the need for tolerating certain design inconsistencies.

Consistency Rules

UML modelling involves many diagrams that are made up of standard notions. The diagrams include Use Case, Class, Activity, State Machine, Sequence, Collaboration, and so on. UML modelling tools generally support drawing UML diagrams without consistency as well. For instance, they might support drawing a sequence diagram with class instances for which actual classes do not exist in the model. Thus it is very important to focus on design models made up of UML for detecting inconsistencies. Unidentified inconsistencies in software design models can obstruct the desired time-to-market attribute of software. The consequences of them are far reaching besides being reflected in the implementation. To overcome this problem, consistency rules can be defined and applied to model-driven software development process. Consistency rules are defined in many research papers. However, we consider the rules explored in [1] for our experiments. The rules are briefly described in Table 1. Before looking at the rules, different kinds of inconsistencies can be visualized in Fig. 1.

As shown in Fig. 1, it is evident that there is no class by name **C**. However, an instance of C is used in the sequence diagram. In the same fashion, the class C instance is used in sequence diagram of Fig. (b) which even does not reflect in the collaboration. Instead, the collaboration diagram shows an instance of class D. These are clearly reflecting model inconsistencies. These inconsistencies in the modelling are quite common when novice users participate in modelling. Unfortunately UML modelling tools are allowing such inconsistencies to be drawn.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

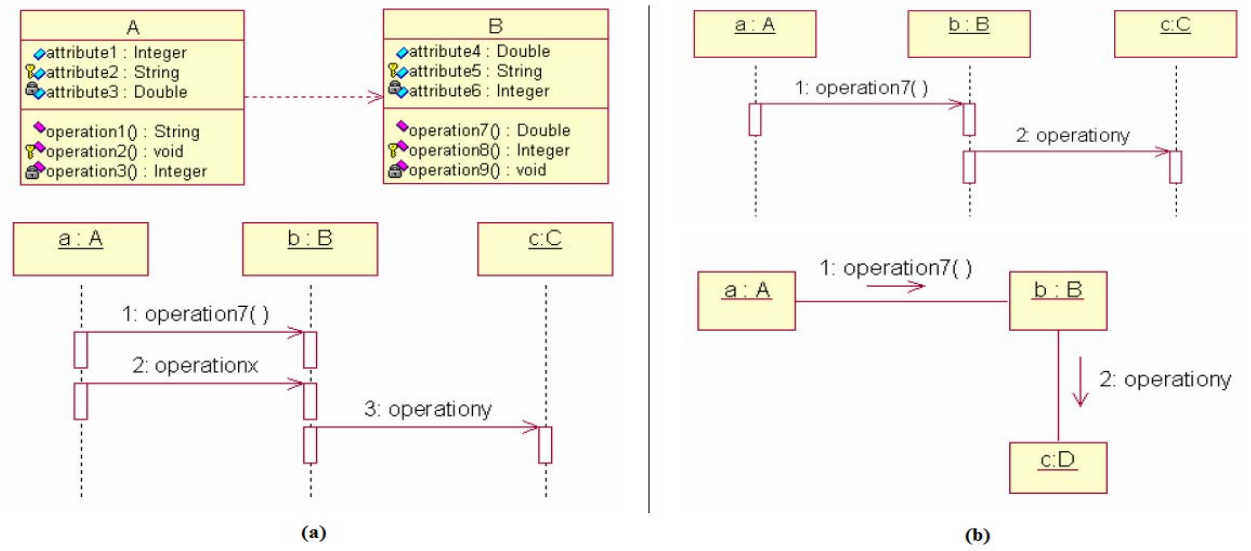

Fig. 1 – Inconsistencies between class and sequence diagrams (a); and between sequence and collaboration diagrams (b)

Table 1 – Consistency Rules

Rule 01	An object in the sequence diagram should exist as a concrete class in class diagram.	Class vs. Sequence
Rule 02	When a class name is modified in class diagram, it should reflect in all instance of sequence diagram synchronously.	Class vs. Sequence
Rule 03	When an object sends message to another object, there must be dependency relationship between them and there must be at least one message between such classes.	Class vs. Sequence
Rule 04	In sequence diagram a message should have corresponding operation in the receiver and it should be visible to sender.	Class vs. Sequence
Rule 05	When an object is deleted from a class diagram, its instances should be removed automatically from sequence diagrams.	Class vs. Sequence
Rule 06	An object represented in sequence and collaboration diagrams should correspond to same class in class diagram.	Sequence vs. Collaboration
Rule 07	An object represented in state machine must be an instance of concrete class in class diagram.	Sequence vs. Collaboration
Rule 08	When a class is deleted from class diagram, corresponding state machine diagrams should be deleted automatically.	Sequence vs. Collaboration
Rule 09	A state represented in state machine diagram should be a legitimate value of an attribute of corresponding class in class diagram.	Sequence vs. Collaboration
Rule 10	The operation used in the state machine diagram should be consistent with the operation in the class diagram in all aspects.	Class vs. State Machine
Rule 11	An activity in state machine diagram must be a message represented in the sequence diagram.	Sequence vs. State Machine
Rule 12	Use cases represented in use case diagram should be reflected in the operations of class diagrams.	Use case vs. Class
Rule 13	Activities and swim lanes in an activity diagram must have corresponding operations in respective classes.	Activity vs. Class

There are four methods, as explored in [1] for consistency checking. They are manual check, compulsory restriction, automatic maintenance and dynamic check. Compulsory restriction does mean that UML modelling tool does not allow invalid design. Automatic maintenance does mean that the modelling tool makes changes automatically to conform to user initiated changes. Dynamic check means the modelling tool can capture user operations in real time and detect inconsistencies. Rule 1, rule 4, rule 6, rule 10, and rule 11 are best applied under compulsory restriction. Rule 7, rule 9, rule 12 and rule 13 are best enforced through manual check. Rule 2, rule 4, rule 5, and rule 8 are ideally enforced through automatic maintenance while dynamic check method is best employed for rule 6, rule 10 and rule 11. These are optimum methods though multiple methods can be employed to enforce a rule.

Tolerance of Inconsistencies

In the wake of the notion "it is possible living with inconsistencies" explored in [2], [3] and [5], our framework also strives to include support for this notion to the extent feasible. This support is also missing in many research papers including the recent one [4]. Sometimes it is essential to ignore some model inconstancy and complete the model element in order to view the proposed concept.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Thus the flexibility of model checking tools to support inconsistencies (reminding that it should be for temporarily) so as to read the benefits in temporary basis. However, we are not advocating tolerance of inconsistencies in any stretch of imagination. We only intend to have temporary support for having limited what if analysis or something of that sort. We believe that the notion of "living with inconsistencies" has to be seen on temporary basis and thus the proposed tool can have a little bit flexibility towards violating consistency rules if the software engineers desire so without having this provision forcibly.

IV. FRAMEWORK FOR EXTENSIBLE REAL TIME SOFTWARE DESIGN INCONSISTENCY CHECKING

Automatically detecting and tracking inconsistencies in software design models has been given importance in research and academia. The model proposed by Egyed [1] is one of the recent efforts to detect design inconsistencies automatically. However, this model can be further improved in terms of personalized configuration and visualization besides making it much more flexible. Our architecture shown in Fig. 2 is aimed at providing user the ability to choose design tool, consistency rule language and visualization preferences. This makes the proposed architecture very flexible and scalable. Our architecture is named eXtensible Real Time Software Design Inconsistency Checker (XRTSDIC) which is designed to be flexible, scalable and extensible with real time response to the model dynamics with regard to detection of inconsistencies.

Since developers work in collaborative fashion with different skill sets, it is possible that they might prefer different notations for modelling. They might use different language to define consistency rules and have different requirements for visualization. The architecture is designed keeping these in mind. The XRTSDIC provides flexibility to have personalized preferences in terms of modelling tool selection, visualization and consistency rule language selection. Based on this architecture, we built a tool that demonstrates the proof of concept. This tool makes the life of developers easy.

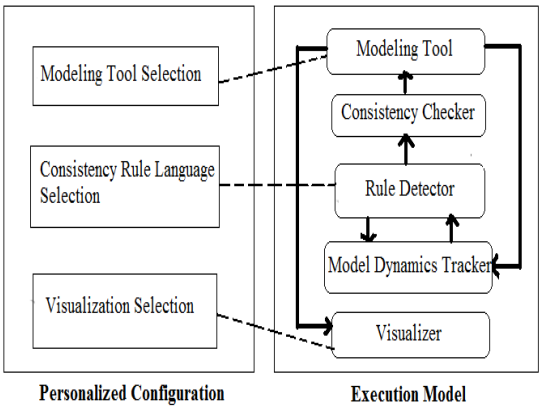

Fig. 2 – Overview of proposed framework

Personalized configuration is the ability of the proposed architecture that lets software engineers to choose the design model they need besides selecting the language for consistency rule making and visualization technique. The execution model will be at work once user chooses preferences for design model, language for consistency rules and appropriate visualization technique. These preferences are personalized and they are automatically made available to the execution model. In the execution model, real time inconsistency verification and visualization of them is made. The process is as described here. As software engineers draw design models, the real time feedback is expected to be given to them. As the model is built, the model dynamics tracker is at work to record the changes being made to the design. The rule detector module takes the model dynamics at runtime and lets consistency checker know the details and rules. The consistency checker does its job to verify consistency and lets the modelling tool to know it and the same is visualized using the visualizer module.

V. PROTOTYPE IMPLEMENTATION

A prototype application is implemnted using Java programming langauge. SWING API is used for user-friendly interface. The implementation has interface for both personalized configurations and actual modeling activities. With respect to personalized configuration, the Configuration menu has provision to view Current Preferences… and Change Preferences… The Start Modeling… option under Model Driven SE invokes the modeling tool based on the user preferences.

As shown in Fig. 3, the application has menu-driven interface that allows users to have personal preferences besides changing them if required. Then the users of the application can start modeling any system desired with specified modeling tool, rule definition language and visualization tool. Out of all the modeling tools that can be conFig.d and used though the prototype application, the default modeling tool is the one we built to demonstrate the proof of concept.

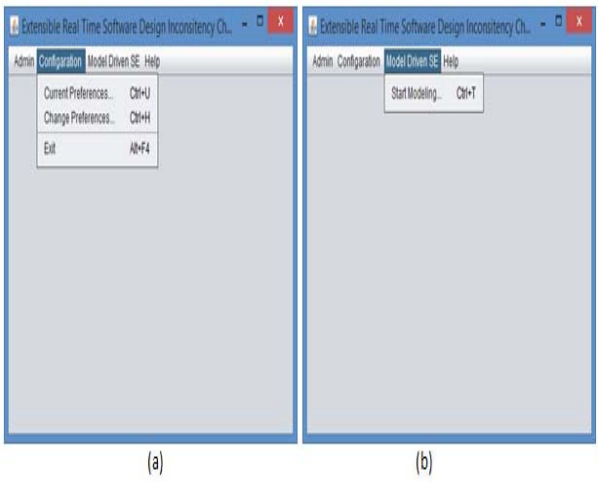

Fig. 3 – Menu-driven UI of the prototype

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

However the "Tracking Inconsistencies" module can be integrated with any modeling tool thus making it very flexible.

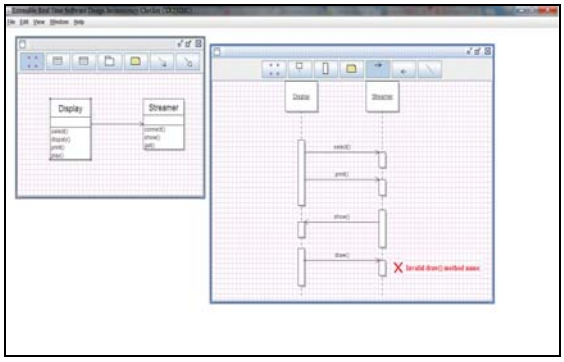

Fig. 4 – The default modelling tool with inconsistency presented

On choosing Start Modelling... in the main prototype application, the default modelling tool with interface as shown in Fig. 4 is presented. Here the end users can perform various modelling operations. The modelling tool supports the 13 rules specified earlier in this paper. The implementation is made in "Tracking Inconsistencies" module that contains classes which encapsulate various functionalities of execution model of the proposed framework XRTSDIC. The functionalities include model dynamics tracking, rule detection, consistency checking, and visualization. The ModelDynamicsTracker class is responsible to monitor real time changes in the model and inform the rule detector. The RuleDetector class encapsulates the functionalities of detection of rules to be applied. The rules to be applied are to be known to ConsistencyChecker class which checks inconsistencies based on the defined rules and informs ModelingTool class of any violations. ModelingTool class encapsulates a modelling tool. The whole process is cyclic in nature and the checking of design inconsistencies is done real time as model elements are being constructed. After obtaining violations, the ModelingTool class invokes Visualizer class to show the inconsistencies with presentation that is in tune with the visualization tool preferred by the user.

The Tracking Inconsistencies module is the common API built as part of the prototype application that can be integrated with any open source modelling tool built in Java. Thus our prototype is capable of supporting ArgoUML, UMLet, UML Designer, and our own tool (default). The prototype has provision to add new modelling tool support in future thus making it unique and flexible which caters to the diversified needs of workforce in software engineering domain.

VI. EXPERIMENTAL RESULTS

The prototype application was used for experiments. The user preferences are used in the execution model of the framework. As per user preferences, the modelling tool is loaded and language for consistency rules is set besides following the chosen visualization method. The proposed framework is tested with default modelling tool and ten UML models. The models are shown in Table 2.

Table 2 – Models used for experiments

Model Name	Class Diagram	Sequence Diagram	State chart Diagram	# Model Elements
ATM	Yes	Yes	Yes	145
Video on Demand	Yes	Yes	Yes	46
Online Courses	Yes	Yes	Yes	185
Billing System	Yes	Yes	Yes	230
Hospital Management	Yes	Yes	Yes	540
Hotel Management	Yes	Yes	Yes	890
University Portal	Yes	Yes	Yes	1230
Defect Tracking System	Yes	Yes	Yes	450
Valuation Portal	Yes	Yes	Yes	1125
School Management	Yes	Yes	Yes	1500

As consistency checking feasibility depends on computational cost utilization of resources, we performed validation with the 10 models listed in Table 2. The models were evaluated with consistency rules pertaining to class, sequence and state chart diagrams. The rule detector plays a vital role in identifying the consistency rule that needs to be evaluated based on the model change. As the model evaluation takes place with an intelligent approach, the model evaluation time decreases significantly. We say that it is an intelligent approach as it uses a hierarchical and incremental approach with the help of accumulated heuristics. This will also improve scalability and accuracy. This is achieved with an incremental and heuristic approach that can eliminate the unnecessary verifications.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

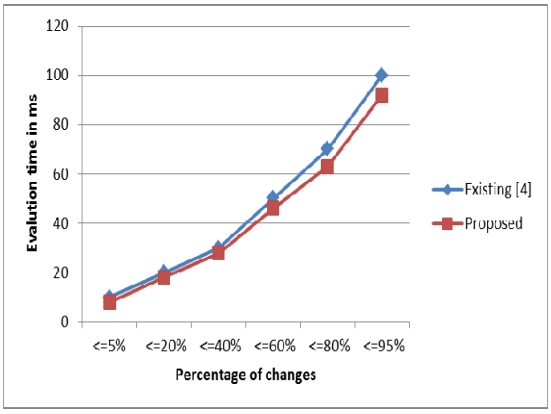

Fig. 5 – Performance comparison

The evaluation time for model changes is presented in Fig. 5. The evaluation time is computed with different percentages of model changes. The average of 10 experiments at each percentage of changes is considered. The results revealed that the proposed approach to evaluate model changes is comparable with the approach followed in [4]. As the results reveal, the evaluation time is negligible and thus the system can run in scalable fashion for large models as well. From the experiments, it is understood that the memory cost linearly increases as model size increases.

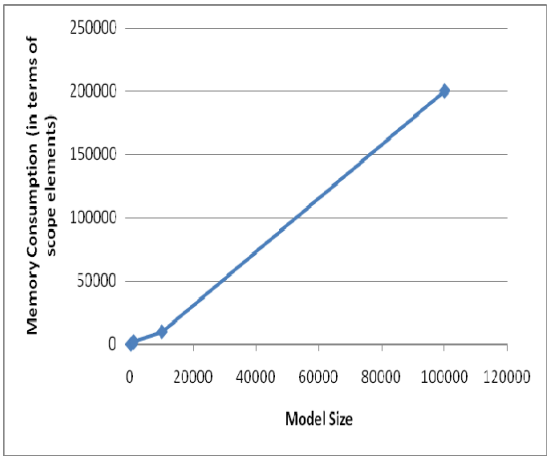

Fig. 6 – Model size vs. memory consumption

As shown in Fig. 6, it is evident that the model size has influence in the memory consumption. However, considering availability of main memory in the modern computers, the memory consumption is not an issue. The memory consumption is involved as the approach is heuristic and incremental model instead of batch processing model. This will affect scalability to some extent. Nevertheless, the proposed system is still scalable as the model changes affect only the rules that are to be evaluated and no unnecessary processing is done.

VII. CONCLUSIONS AND FUTURE WORK

Detecting and tracking inconsistencies in the software design models can help software engineers to unearth bugs early in the life cycle of the system. This will result in the optimization of time, cost and effort required to complete projects successfully. In this paper we studied existing tools and approaches that are used to check inconsistencies in UML design models. Most of the existing approaches were focusing on different aspects of consistency checking using certain techniques. However, a holistic and comprehensive approach is missing. Since UML modelling tools do not provide consistency checking features, it is inevitable to have consistency rules to be defined. There are many modelling tools available and the software engineers may choose any one of them. In this paper we proposed a framework that facilitates software engineers to choose a modelling tool from a set of tools, and to choose a language for specifying consistency rules and a method for visualization. To achieve this framework has two things such as personalized configuration and execution model. The former allows users to determine preferences with respect to modelling tool, language for consistency rules and a visualization method while the latter takes care of detection and tracking of consistencies in the design models. We built a prototype application that demonstrates the flexible and real time consistency checking in UML models. Our empirical results revealed that there is significant improvement in speed, accuracy and scalability in the proposed model. Moreover it is extensible with other models, rules, visualization methods. This research can be extended further to enhance the proposed framework with support for more consistency rule languages, visualization methods and modelling tools.

REFERENCES

[1] Xianhong Liu, Identification and Check of Inconsistencies between UML Diagrams. Journal of Software Engineering and Applications. p1-5, 2013.

[2] W. Shen, K. Wang, and A. Egyed, “An Efficient and Scalable Approach to Correct Class Model Refinement,” IEEE Trans. Software Eng., vol. 35, no. 4, pp. 515-533, July/Aug. 2009.

[3] R.N. Taylor, R.W. Selby, M. Young, F.C. Belz, L.A. Clarce, J.C.Wileden, L. Osterweil, and A.L. Wolf, “Foundations of the Arcadia Environment Architecture,” Proc. Fourth Symp. Software Development Environments, ACM, 1998.

[4] Alexander Egyed, “Automatically Detecting and Tracking Inconsistencies in software Design Models. *IEEE*. 37 (2), p188-204, 2011.

[5]. Bashar Nuseibeh, Steve Easterbrook, Alessandra Russo. (2001). Making inconsistency respectable in software development. *Elsevier*. p1-5, 2001.

[6]. Christian Nentwich, Wolfgang Emmerich and Anthony Finkelstein, Static Consistency Checking for Distributed Specifications.*Research Paper*. p1-10, 2000.

[7]. Tom Mens, Ragnhild Van Der Straeten, and Maja D’Hondt, “Detecting and Resolving Model Inconsistencies Using Transformation Dependency Analysis”, *Springer*. p1-15, 2006.

[8]. Reiko Heckel, Jochen Kuster, Gabriele Taentzer, “Towards Automatic Translation of UML Models into Semantic Domains”, *Research Paper*. P1-5, 2000.

[9]. Ken Kaneiwa and Ken Satoh, “Consistency Checking Algorithms for Restricted UML Class Diagrams”, *Research Paper*. p1-21, 2000.

[10]. Masters thesis Torger Kielland, “Consistency checking of UML models on the XMI format”, *Research Paper*. p1-83, 2000.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

[11]. Bogumila Hnatkowska, Zbigniew Huzar, Jan Magott, "Consistency Checking in UML Models", *Research Paper*. p1-6, 2000.
[12]. Kenneth Baclawski, Mieczyslaw M. Kokar, "Consistency Checking of Ontologies Expressed in UML.", *Research Paper*. p1-9, 2000.
[13]. Zs. Pap, I. Majzik1, A. Pataricza and A. Szegi, "Completeness and Consistency Analysis of UML Statechart Specifications", *Research Paper*. p1-15, 2000.
[14]. Ali Hanzala Khan, Ivan Porres, 'Consistency of UML class, object and statechart diagrams using ontology reasoners", *Elsevier*. p4-5, 2015.
[15]. Ragnhild Van Der Straeten, Jocelyn Simmonds, "Detecting Inconsistencies between UML Models Using Description Logic", *Research Paper*. p1-9, 2013.
[16]. Jocelyn Simmonds and M. Cecilia Bastarrica, "Description Logics for Consistency Checking of Architectural Features in UML 2.0 Models", *Research Paper*. p1-15, 2005.
[17]. Xiangpeng Zhao, Quan Long, and Zongyan Qiu, "Model Checking Dynamic UML Consistency", *Research Paper*. p1-20, 2005.
[18]. Zinovy Diskin, Yingfei Xiong, and Krzysztof Czarnecki, "Specifying Overlaps of Heterogeneous Models for Global Consistency Checking", *Research Paper*. p1-15, 2005.
[19]. Alexander Egyed, "Instant Consistency Checking for the UML.", *ACM*. p1-5, 2006.
[20]. Clare Gryce, Anthony Finkelstein and Christian Nentwich, "Lightweight Checking for UML Based Software Development", *Research Paper*. p1-9, 2006.
[21]. Francisco J. Lucas , Fernando Molina , Ambrosio Toval, "A systematic review of UML model consistency management", *Elsevier*. p1-10, 2009.
[22]. Wafa Chama, Raida Elmansouri and Allaoua Chaoui., "Model checking and code generation for uml diagrams using graph transformation", *International Journal of Software Engineering & Applications*. 3 (6), p1-17, 2012.
[23]. Xavier Blanc, Isabelle Mounier, Alix Mougenot and Tom Mens, "Detecting Model Inconsistency through Operation-Based Model Construction", *ACM*. p1-9, 2008.
[24]. Yann Thierry-Mieg Lom-Messan Hillah, "UML Behavioral Consistency Checking using Instantiable Petri Nets", *Research Paper*. p1-5, 2006.
[25]. Jean Louis SOURROUILLE, "A Pragmatic View about Consistency Checking of UML Models.", *Research Paper*. p1-8, 2006.
[26]. R. Dubauskaite, O. Vasilecas., "Method on Specifying Consistency Rules among Different Aspect Models", expressed in UML.*ISSN*. 19 (3), p1-5, 2013.
[27]. Alexander Egyed, "Scalable Consistency Checking between Diagrams – The VIEWINTEGRA Approach", *IEEE*. p1-4, 2001.
[28]. JOCELYN SIMMONDS, "A tool based on dl for uml model consistency checking.", *Research Paper*. p1-5, 2005.
[29]. Alexander Egyed, "UML/Analyzer: A Tool for the Instant Consistency Checking of UML Models", ICSE, P1-5, 2007.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)