World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:3, 2009

A Tool for Checking Conformance of UML Specification

Rosziati Ibrahim, and Noraini Ibrahim

***Abstract*—**Unified Modeling Language (UML) is a standard language for modeling of a system. UML is used to visually specify the structure and behavior of a system. The system requirements are captured and then converted into UML specification. UML specification uses a set of rules and notations, and diagrams to specify the system requirements. In this paper, we present a tool for developing the UML specification. The tool will ease the use of the notations and diagrams for UML specification as well as increase the understanding and familiarity of the UML specification. The tool will also be able to check the conformance of the diagrams against each other for basic compliance of UML specification.

***Keywords*—**Software Engineering, Unified Modeling Language (UML), UML Specification.

I. Introduction

SOFTWARE development life cycle (SDLC) is used to process the activities of software development. Four main phases are used in SDLC. There are analysis, design, implementation and testing [5]. In SDLC, modeling tool is usually used to do the analysis of a system. The modeling tool used can be either a structured approach or an object-oriented approach or a hybrid approach. A structured approach uses diagrams such as entity relationship diagrams (ERD) and context diagrams to model and analyze the system requirements. Object-oriented approach, on the other hand, uses diagrams such as use-case diagrams and class diagrams to model and analyze the system requirements. A hybrid approach is a combination of a structured and object-oriented approach.

Unified Modeling Language (UML) is one of the modeling tools that are often used for object-oriented approach. UML assumes a process that is use-case driven, architecture-centered, iterative and incremental [1]. UML is a standard language for visually describing the structure and behavior of a system [8]. Therefore, during analysis, system requirements are transformed into UML specification using diagrams. These diagrams are Use-Case Diagram, Class Diagram, Interaction Diagram, Communication Diagram, Activity Diagram, State Diagram, Component Diagram and Deployment Diagram [7]. These diagrams have special notations that someone has to be familiar with them in order to use the notations.

The foundation of a good application begins with a good analysis. In order to do a good analysis, we should be able to rigorously analyze the system requirements with the help of the tool. Therefore, for educational purpose and for ease of use and modeling, we propose a tool for developing the UML specification. Modeling is an essential part of any projects. A model plays a major role in system development life cycle which serves as a blueprint for the system. This tool can be used for specifying the notations and developing the UML specification diagrams. The tool can also be used to check the conformance of the diagrams against each other for basic compliance of the UML specification. The purpose of producing the tool is to ease the user in understanding the notations and diagrams in UML specification as well as used them for the requirements specification and checking conformance of the diagrams. That is, once the diagrams have been finalized, the tool can be used to check whether the diagrams comply with the syntax, rules and notations imposed by the UML specification.

The rest of the paper is organized as follows. Section II reviews the UML Specification and Section III presents the related work. Section IV discusses our tool in details, in particular on how to use the notations and develop the diagrams using the tool. The conformance of the diagrams is also discussed in Section IV. Finally, we conclude our paper in Section V and give some suggestions for future work of the tool.

Authors are with Faculty of Information Technology and Multimedia, Universiti Tun Hussein Onn Malaysia (UTHM), Batu Pahat, 86400, Johor, Malaysia (e-mail: rosziati@uthm.edu.my, noraini@uthm.edu.my).

II. Review of the UML Specification

Requirements analysis is an important phase during the development life cycle. In UML specification, requirements analysis is usually done using diagrams [8]. Most often, during analysis phase, system requirements are transformed into UML specification using diagrams. These diagrams have special rules and notations.

A use-case diagram is used to specify requirements of the system. In a use-case diagram, two important factors are used to describe the requirements of a system. They are actors and use cases. Actors are external entities that interact with the system and use cases are the behavior (or the functionalities) of a system [9]. The use cases are used to define the requirements of the system. These use cases represent the functionalities of the system. In most cases, use cases are developed based on the user perspective since the user is going to use the system.

In this paper, for the ease of explanation about the UML specification, we present an example of an application for monitoring system of a postgraduate student submitting his/her progress report to Centre of Graduate Studies. For this system, during the requirements analysis, we have to capture the requirements of the system. The requirements of the system include the capability to submit progress report using the provided form, view the submitted progress report and evaluate the submitted progress report. These three

requirements are then transformed into a use-case diagram as shown in Fig. 1.

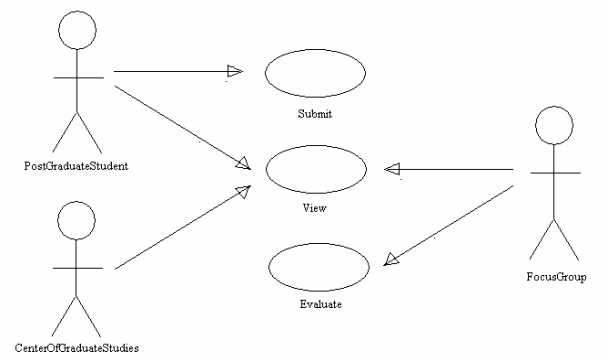

Fig. 1 A Use-case Diagram for
Monitoring System of Postgraduate Student

Fig. 1 shows a simple use-case diagram for a monitoring system of postgraduate student where a postgraduate student (an actor) can submit his/her progress report to Centre of Graduate Studies. From Fig. 1, a student is able to do two tasks: submit a progress report and view a progress report. A focus group is able to view and evaluate the progress report while the centre is able to view the progress report.

Most often, use cases represent the functional requirements of a system. If the requirements are gathered correctly, then a good use-case diagram can be formed. Once the use-case diagram is formed, the interaction diagram can then be developed. In UML specification, interaction diagrams are usually used to manually record the behavior of a system by viewing the interaction between the system and its environment [7]. These interaction diagrams describe in details activities for use cases. Fig. 2 shows the interaction diagram for use case *Submit* for a scenario when an incomplete progress report is submitted.

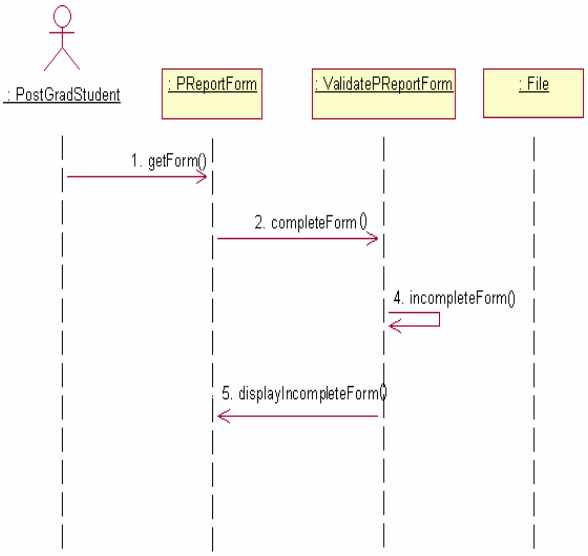

Fig. 2 Interactive Diagram for Submit when an incomplete form is Submitted

Once the use-case diagram and interactive diagram are formed, the next diagram, an activity diagram can be developed. An activity diagram, on the other hand, describes the activities of the process. The purpose of an activity diagram is to provide a view of flows and what is going on inside a use case [1]. Fig. 3 shows an example of an activity diagram which exhibits the activities that can be performed by a postgraduate student. From a use-case diagram in Fig. 1, a postgraduate student is able to submit and view the progress report. Hence, the activity diagram shows that these two activities can be performed by the postgraduate student.

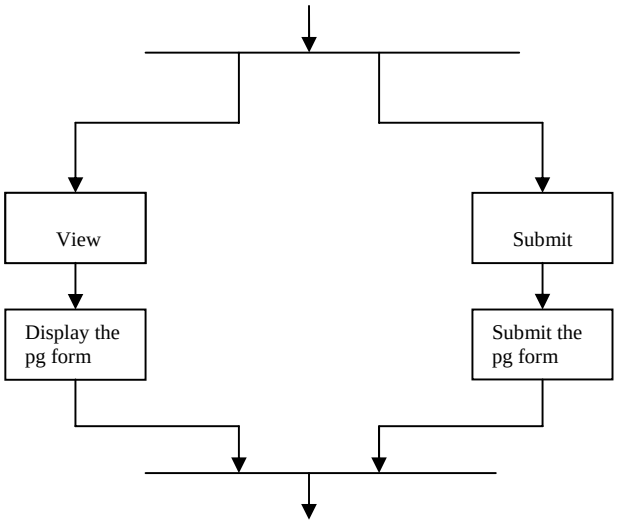

Fig. 3 An Activity Diagram for Postgraduate Student

The next diagram that can be developed after the formation of use-case diagram, interaction diagram and activity diagram is the class diagram. The class diagram is the main static analysis diagram [1]. It shows the static structure of the model for the classes and their relationships. They are connected to each other as a graph. Each class has its own internal structures and its relationships with other classes. Fig. 4 shows an example of a class diagram for Monitoring System of Postgraduate Student.

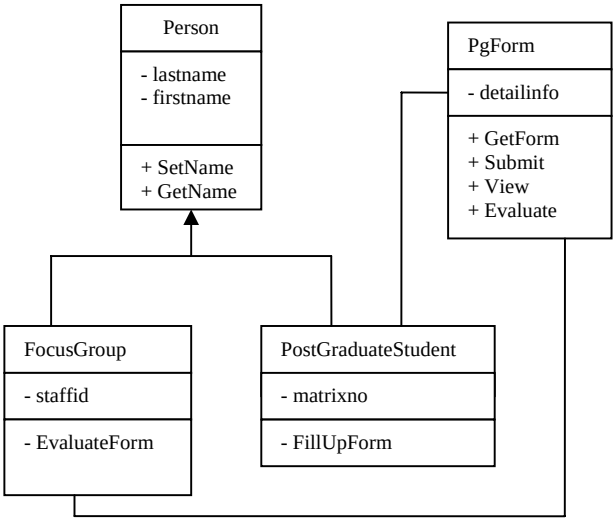

Fig. 4 A Class Diagram for Monitoring System of Postgraduate Student

From Fig. 4, each class consists of a class name, its attributes and methods. For example, a class *Person* has attributes lastname and firstname with methods SetName and GetName. Classes *FocusGroup* and *PostgraduateStudent* inherit class *Person*. Class *FocusGroup* declares its own attribute (staffid) and one method (EvaluateForm) and class *PostGraduateStudent* declares its own attribute (matrixno) and one method (FillUpForm). Note that, a subclass inherits all the attributes and methods of its superclass. Class *PgForm*, on the other hand, offers 4 methods namely GetForm, Submit, View and Evaluate. Table I shows the mapping from use-cases from Fig. 1 into functions in the class diagram in Fig. 4. This mapping is important for the consistency of the UML diagrams.

TABLE I
USE CASES MAPPING TO SYSTEM'S FUNCTIONALITIES

Use Case	Function
Submit	Submit
View	View
Evaluate	Evaluate

Once the use-case diagram, interaction diagram, activity diagram and class diagram have been formed, we have to check the consistency of these diagrams with each other. Based on the use-case diagram, interaction diagram, activity diagram and class diagram, a basic compliance of UML specification can be derived. Compliance means complying with its abstract syntax, well-formedness, semantics and notations [12]. Compliance can be formed using UML metamodel. The UML metamodel for class diagram, for instance, consists of an abstract syntax of class diagram, a set of well-formedness rules that define the abstract syntax of the UML class diagram and informal descriptions of semantics.

III. RELATED WORK

UML is a modeling language for object-oriented approach. Many developers use UML to develop the system specification prior to implement the system. However, the developers have to be familiar with the set of notations imposed by the UML specification. Many tools are available in market to help the developers to develop the UML specification, for example, Visio [4], Cadifra [2] and Rational Rose [9].

Visio tool [4] allows the user to create UML specification diagrams. The tool is used for visualization of the model only. Cadifra tool [2] is used for UML Editor. The tool allows a user to draw the UML specification diagrams for Window-based environment. Rational Rose tool [9] is used for specifying activities and processes for software development. During developing the software requirements document (SRD), the tool can be used to draw the UML specification diagrams.

What is lacking in Visio, Cadifra and Rational Rose tools is that they are unable to check the compliance of the UML specification. For our tool, instead of just drawing and developing the UML specification diagrams, the tool is able to check the conformance of the diagrams. That is, the basic compliance of the UML specification diagrams can be done using the tool. This will ensure the diagrams developed adhere to the set of rules and notations imposed by the UML specification.

Checking compliance of the UML specification is an ongoing research. France et al. [3], for example, use UML-based pattern specification technique for checking the compliance of a pattern design specification. Weimer et al. [13], on the other hand, use specification mining for learning the program specifications. Alexander Egyed [4], for instance, develops a UML analyzer tool for checking the consistency of UML diagrams. The UML analyzer is a system for defining and analyzing the conceptual integrity of UML models by taking the concept of software development, which is about modeling a real problem, solving the model problem, and interpreting the model solution in the real world [4]. In doing so, a major emphasis is placed on mismatch identification and reconciliation within and among system views (such as diagrams). UML analyzer describes and identifies causes of architectural and design mismatches across UML views as well as outside views represented in UML. The UML analyzer tool is quite similar to our tool. However, UML analyzer uses Java programming and concentrates on architecture and design mismatches across UML view. Our tool, on the other hand, uses C++ programming and concentrates on UML diagrams and proof the soundness of the diagrams using the diagrams abstract syntax, its well-formedness and semantics, and notations of the UML Specification.

IV. THE UMLST

The tool, which we call UMLST (Unified Modeling Language Specification Tool), can be used to develop the UML specification diagrams from any system requirements. The tool is also able to check the conformance of the developed diagrams against each other if all the diagrams are created within one project. The tool is developed using object-oriented approach with C++ programming language. The tool has 3 major components with 2 stages as shown in Fig. 5. Stage 1 allows a user to use the tool as an Editor to develop the UML specification diagrams using the Workspace provided. Stage 2 allows a user to use the Engine of the tool to group the diagrams that have been drawn in Stage 1 for checking the conformance of the diagrams. The engine of the tool then takes these diagrams and checks the conformance of the UML specification diagrams drawn.

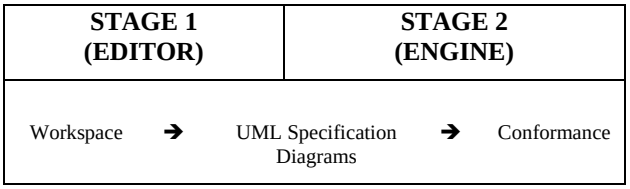

Fig. 5 The Components of UMLST

From Fig. 5, the tool allows a user to develop the UML specification diagrams of any system in the workspace provided. The workspace is used as a place for a user to provide the system requirements by means of the diagrams. In the workspace, a ToolBox is used to create, edit and display the diagrams. The ToolBox consists of standard symbols and notations for each of the diagrams used. For example, for use-

case diagram, symbols for an actor and a use case, and arrows for connecting an actor with use cases can be used. In the Workspace, a user can also type-in the text for each of the use cases used by using the Edit command in the Menu Bar provided by the tool. The Workspace will allow a user of the tool to develop the use-case diagram according to any system's requirements. Fig. 6 shows the user interface design of the tool. The diagrams can also be saved for later use and printed for hardcopy.

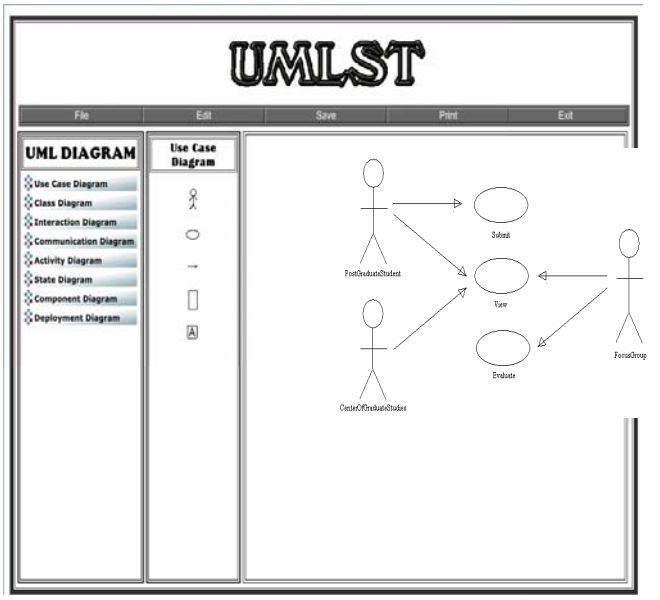

Fig. 6 The Interface Design of UMLST

If a user wants to draw another diagram, such as a class diagram, then the option class diagram can be selected and the ToolBox for the class diagram will appear. Note that the ToolBox is designed in such a way that the correct notations and symbols will only appear for the correct diagrams.

Once all the UML specification diagrams have been finalized, the user can check the compliance of the diagrams for the second stage of the tool.

For the second stage, in the File command in the Menu Bar, the diagrams can be grouped together to check the compliance of the diagrams. For the basic compliance, the class diagram, use-case diagram, interaction diagram and activity diagram are grouped and extracted from the project. Then, all the diagrams are checked one by one against the set of rules and notations imposed by the UML specification which are stored inside the tool database for conformance. The checking consists of two main activities. Firstly, the diagrams are checked against each others for any mismatch words. Table II, for example, shows that the use-case diagram is checked against other diagrams (class, activity and interaction) for mismatch words.

TABLE II
CONSISTENCY CHECK OF UML DIAGRAMS

Diagram	Check with other Diagram
Use-Case	Class
Use-Case	Activity
Use-Case	Interaction

From Table 2, the functionalities of the system are checked using the words used for declaring use cases in use-case diagram and functions in class diagram. Each use case is transformed into token. If the tool finds any mismatch words, the tool will abort indicating the inconsistency diagrams. Then, the actors used in the use-case diagram are checked using the activity diagrams declared for each actor. The use cases used for each actor in the use-case diagram is checked against the activities declared in activity diagram for each actor. Again, if the tool finds any mismatch words, the tool will abort indicating the inconsistency diagrams. Then, the use-case diagram is checked with the interaction diagrams declared for each use cases declared in use-case diagram.

Once the mismatch words are checked, the second activity for the engine is to check the soundness of the diagrams using the diagrams abstract syntax, its well-formedness and semantics, and notations of the UML specification. For this activity, a set of inference rules are used to establish the well-formedness of the diagrams. Theorem prover is also used for the soundness of the diagrams.

V. CONCLUSION AND FUTURE WORK

UMLST is a tool that is able to develop the UML specification diagrams according to the system requirements. The diagrams can then be checked for conformance. The purpose of UMLST is to develop the UML specification diagrams and check its conformance. Modeling is an important part of any projects. A model plays a major role in system development life cycle. A model is also served as a blueprint for the system.

Currently, UMLST is able to check the conformance of the diagrams for basic compliance using class diagram, activity diagram, interaction diagram and use case diagram. We intend to extend UMLST so it is able to perform a complete compliance using the reminder of the diagrams as well as any advance features offered by UML specification.

ACKNOWLEDGMENT

The authors would like to thanks Universiti Tun Hussein Onn Malaysia (UTHM) for supporting this research under the short term research grant.

REFERENCES

[1] Bahrami A. (1999). Object-Oriented Systems Development, Mc-Graw Hill, Singapore.
[2] Cadifra UML Editor. (2008), http://www.cadifra.com/
[3] France R., Kim D.K., Ghosh S., and Song E. (2004). A UML-Based Pattern Specification Technique, IEEE Transactions on Software Engineering, Vol. 30, No. 3, March 2004.
[4] Egyed A. (2008). UML Analyzer Tool, http://www.alexander-egyed.com/tools/uml_analyzer_tool.html
[5] Hoffer J., George J. and Valacich J. (2008). Modern Systems Analysis and Design, 5th Edition, Pearson International Edition, New Jersey.
[6] Microsoft Visio Toolbox. (2008), http://www.visiotoolbox.com/
[7] Miller G. (2003). What's New in UML 2.0, A Borland White Paper, http://www.borland.com/
[8] OMG. (2004). OMG Unified Modelling Language (UML) Superstructure Specification, http://www/omg.org/
[9] Rational. (2003). Mastering Requirements Management with Use Cases, Rational Software, IBM.
[10] Rational Rose (2008), http://www.rational.com/

[11] Sommerville I. (2007). Software Engineering, 8th Edition, Addison Wesley, England.
[12] UML. (2004). UML 2.0 Infrastructure Specification, http://www/omg/org/docs/ptc/03-09-15.pdf
[13] Weimer W. And Mishra N. (2008). Privately Finding Specifications, IEEE Transactions on Software Engineering, Vol. 34, No. 1, Jan/Feb 2008.