From Modeling to Code Generation: An Enhanced and Integrated Approach

Oluwasefunmi Tale Arogundade, Olutimi Onilede, Sanjay Misra, Olusola Abayomi-Alli, Modupe Odusami, and Jonathan Oluranti

Abstract

Information system drives every aspect of human endeavor, and it is a major stakeholder in human existence. Systems with poor modeling suffer a lot from poor implementation down to poor performance due to lack of critical subjection and testing. Software modeling is, therefore, of paramount importance in order to achieve a reliable system. There has been a lot of works done in software modeling, and eventually, the Universal Modeling Language was formulated to create a standard for software modeling. Although there have been some development or modeling tools that can be used to model a software system and the design then converted to software codes that can then be perfected, none of these tools has considered security and integrated as a single tool. Therefore, this paper focuses on building an integrated system (all-encompassing system) for building UMLsec-based modeled systems that will convert UML diagrams to code. The system integrates Eclipse Mars incorporated with Papyrus modeling plug-ins and Eclipse Kepler with Java EE incorporated with CARiSMA plug-ins. These four tools were integrated together by an executable application built with NetBeans. The system was tested by modeling an e-government system from the class diagram to analysis and code generation.

Keywords

Information system • Software modeling • UML • UMLsec

1 Introduction

Software modeling is of paramount importance in order to achieve a reliable system (John and Edinbarough 2017). This is one of the areas of enhancing software development process with a focus on tackling complexities (Hovsepyan et al. 2014). A well-defined software development process using appropriate development methods and tools automatically provides high software quality (Wallin 2002). Poorly modeled systems suffer from poor implementation to poor performance due to the lack of critical subjection and testing. There has been a lot of work done in software modeling, and eventually, the Universal Modeling Language was formulated to create a standard for software modeling (Kaur and Singh 2009). The growth of UML recently has helped in the area of communication and documentation with a focus on tasks that do not need a formal semantics tool support (Broy and Cengarle 2011).

With UML standard, there are many challenges and issues about modeling software, particularly, how perfect a model can be translated to the software.

There have been some development or modeling tools that can be used to model a software system and then converted to a software code that can then be perfected, but none of these has been integrated into single automated tool. This paper focuses on building an automated system for building UML-based modeled system that will convert UML diagrams to code and verify such a model.

O. T. Arogundade · O. Onilede
Deparment of Computer Science, Federal University of Agriculture Abeokuta, Abeokuta, Nigeria
e-mail: arogundadeot@amss.ac.cn

O. Onilede
e-mail: oniledeo@babcock.edu.ng

S. Misra (✉) · O. Abayomi-Alli · M. Odusami · J. Oluranti
Department of Electrical and Information Engineering, Covenant University, Ota, Nigeria
e-mail: sanjay.misra@covenantuniversity.edu.ng

O. Abayomi-Alli
e-mail: olusola.abayomi-alli@covenantuniversity.edu.ng

M. Odusami
e-mail: modupe.odusami@covenantuniversity.edu.ng

J. Oluranti
e-mail: jonathan.oluranti@covenantuniversity.edu.ng

P. K. Singh et al. (eds.), *Innovations in Information and Communication Technologies (IICT-2020)*,
Advances in Science, Technology & Innovation,
https://doi.org/10.1007/978-3-030-66218-9_50

Considering existing modeling tools for UML diagrams, there has been none with the capability of converting UML diagrams to programming code and verifying the programmability or compatibility of such code to the programming language to which the diagrams were converted. This problem is impeding the further development in UML such that a solution to it will greatly enhance the software development process. Therefore, this study aims to develop an automated verification system that will enhance some selected modeling platforms in the process of software development. The scope of this study is limited to the automating the verification of UML diagram by converting the class diagram to Java code. The remaining paper is sectioned as: Sect. 2 describes in detail existing studies, while Sect. 3 discusses extensively the proposed system design. Section 4 presents the proposed system methodology, while Sect. 5 explains the implementation and testing, Sect. 6 discusses the results obtained, and the paper concludes in Sect. 7 with future recommendations.

2 Literature Review

An overview of existing literatures by previous researchers is analyzed in this section.

Ciccizzi et al. (2012) gave an efficient implementation from source models based on the combination of Action Language for Foundational UML with UML. Authors further developed an industrial embedded system through exploiting a model-driven engineering approach. The study was able to improve productivity and also combat possible errors relating to code-based approaches. Ringert et al. (2014) introduced a conceptual framework for generating code composition based on MontiArcAutomation. The study was able to effectively integrate generated code in post hoc application-distinct languages for modeling robotics SE. Authors stated the need to widen applications on syntax, approach, and technical solutions. Further subsections discuss the related studies and the comparison of this study to the previous work.

2.1 Related Works

This subsection gives a detailed review of different automated verification tools for UML models.

Arogundade et al. (2018) improved on the design of a reliable UML profile for civil status and rights and validating the model using Object Constraint Language (OCL) and Papyrus. Noyer et al. (2014) employed code generators to generate source code from UML models based on standardized extensible markup language metadata interchange (XMI) format. This method is achieved by conceiving metamodels for application programming model interfaces of UML tools, after which the code generators can produce the source code. Shiferaw and Jena (2018) described an adequate depiction of the UML model as well as the development of an enabling environment for display. The approach identifies system functionality with a platform-independent model and generates executable source code from the model. Thus, the usability of every single object is identified through the UML statechart diagram. Bhullar et al. (2016) presented an approach for code generation using behavioral UML diagrams such as activity diagrams, sequence diagrams, and use case diagrams. The result shows that no method is ideal for giving a 100% code generation. However, some enhancement methods are required to improve the code generation. Yamada (Yamada and Wasaki 2011) presented an approach that models on the design specification of software using formal language and verifies it by the model checker. The approach converts the UML activity diagram automatically into the SPIN model checking code PROMELA. An extension of the SPIN notation PROMELA was used in addition to other primitives for modeling simultaneous object-oriented systems, which comprises class definition, object instantiation, message sent, and synchronization (Jiang 2009). Amirat et al. (2012) presented an automated conversion of UML sequence diagrams with imbricate connected fragment to PROMELA code. This approach is used to simulate the execution and to verify properties written in linear temporal logic with SPIN model checker.

Future studies have shown the tremendous growth in using UML for developing of security-critical systems. A typical instance from Kim et al. (2004) defines a method for specifying role-based access control policies for modeling UML designs (Poniszewska-Maranda 2012; Ray et al. 2004; Cenys et al. 2009). This model allows developers to specify behaviors of violations against the policies. In addition, Breu and Popp (2004) presented an approach for the specification of user rights using UML. The approach is based on first-order logic with a built-in notion of objects and classes with an algebraic semantics and can be realized in Object Constraint Language (OCL). Beckert (Beckert et al. 2002) presented logical reasoning based on UML models using the power of interactive theorem. Franconi et al. (2018) extended the study by Beckert et al. (2002) by identifying and discussing the fragments of OLC using relational algebraic query. Thapa et al. (2010) used the UML metamodel extension mechanism given as profiles to verify whether a UML design model gratifies its domain-distinct security and time-related requirements in an integrated tool environment. UMLsec and MARTE (UML profile for Modeling and Analysis of Real-Time and Embedded systems) were combined into the UML metamodel in tackling both security and timing properties.

Sebastian et al. (2020) investigated model-driven architecture based on scientific evidence by carrying out a systematic literature review on model-driven approaches in software engineering. The authors concluded that the most widely used language for modeling is UML. Sunitha and Samuel (2019) proposed a novel method for the implementation of state diagram comprising hierarchical, concurrent, and history states. Code generation was based on the UML state machine, and experimental results showed that the proposed model is a promising approach compared with other methods.

2.2 Comparison with Our Previous Work

This study is an extended version of previous work on Arogundade et al. (2016) with focus on developing an automated verification system that will enhance some selected modeling platforms in the process of software development. In our previous work, we were able to model with UML and also generated codes but unable to verify the programmability and compatibility of the code generated to Java which was the programming language used. However, this study was able to combine all the various tasks involved from modeling to code generation into a single platform through the development of a simple effective environment. Specifically, the proposed system is expected to automate the process of drawing class diagrams for a system, convert it to Java code, and analyze the diagram in CARiSMA.

To the best of our knowledge, none of the existing studies on binding UML to model checkers has been extended to analyzing UMLsec models. This study is motivated by the need to support security constructs. In addition to the translation of complex data types, which have proven to be an important aspect in supporting cryptography extension.

3 Research Methodology

The aim of modeling software system with UML and developing an integrated tool that convert UML class diagram into readable instructions is to give applicable solution and result which is aided by functional philosophy (Saunders et al. 2019). In order to achieve this result, there are various steps of build and process method derived from design research aspect that should be followed (Amaral 2011). This build and process method differs from design science research methodology strategy. The focus of this research is to design an integrated tool that enables software engineer to model the software using UML class diagram and generate codes from validated class diagram. The process begins with logical induction method to a more understanding of the nature of challenges faced when the need arises for a structured and efficient model that will aid software engineer deliver their software in time without delay (Saunders et al. 2019).

The steps involved in the build and process approach will be enumerated before presenting the detail of the proposed methodology. The building process was launched on the part of framework for computing research methods which is in unison with the functional approach used (Holz et al. 2006). The relevant and useful aspects to this study were given consideration in the next section.

Framework for Computing Research

- What are we trying to establish?
  a. (To search for more knowledge)—the researchers look for more information on modeling of software and analyzing the model for validation which will help in formalization of the model in a way that it is easy for a programmer to read and understand.
  b. (To design an efficient process)—the author will model a software and develop an integrated tool that will convert the class diagram into readable instructions which is comprehensible to software developers.
- Where do we get the needed data from?
  c. (SWOT)—the writers will review the literatures to know the strength, weakness, opportunities, and threats in the area of software modeling, with UML, analysis, and conversion of validated class diagram to readable and comprehensive codes.
  d. (Be attentive, interaction)—the authors need to interact with database/IT officers in e-government service in Nigeria to acquaint themselves to the processes followed in carrying out their day-to-day activities. This step is necessary for a proper software process modeling.
  e. (Model the process)—the authors will use UML to model the e-government domain used for validation.
  f. (Gathering source)—the researchers will get useful and current information from the domains used in this study and from conceptual analysis of the literatures.
- What is the usefulness of the collected data?
  g. (For better understanding of the process)—the authors will use the collected data for better understanding of the processes involved in e-government service for a good process modeling.
  h. (Software modeling, formalization)—the researchers will design UML metamodel using class diagram and formalize with CARiSMA.

- Do the research purpose achieved?
  i. (Verification of result, conclude)—the writers would have gained more knowledge on software modeling, and the conversion of class diagram by now develops a functioning tool. These authors will be able to verify the outcome of the model.
  j. (Future work)—the authors will conclude from the design and process modeling and propose a future direction of the work.

3.1 System Design Considerations

This section shows the overall design of the system model incorporating security, portability, openness, and other important factors that make Java to edge other languages to be the focus of the design in this section. In this paper, the system requirement is divided into a functional and non-functional requirement.

Functional Requirement

These are statements of services the system should provide some of which include what response to anticipate from the system when some distinct inputs are administered and the likely behavior of the system, in distinct directions Sommerville (Sommerville 2011). A functional requirement indicates substance that the developer requires to compile in delivering the solution. They are listed below:

i. The system shall grant the user to close the program when he has finished using the system.
ii. The system shall grant user login based on their username and password.
iii. The system will grant the user to choose the task to perform at any point in time.

Non-functional Requirement

These are system properties and constraints on functions provided, e.g., reliability and response time. One major property of the system is that it is portable; i.e., the system is platform-independent.

Apart from this, the system also has the following constraints:

i. Users must log based on their username and password to use the system.
ii. The password must be the complexity required.

User Requirements

They explain the needs of end-users from a system. They explain the actions that users must be able to carry out. These requirements are:

i. Administrator shall be able to login with their username and password.
ii. Admin must be able to carry out tasks.

3.2 Building the Installer Package

After building the application successfully, there is a need to bundle all the tools together so as to make deployment of the tools on another system to be easy and to eliminate the need for replication of the process of setting up the various components and requirements of the system.

This CARiSMA application was built to extract all the necessary installation files needed in Eclipse Kepler with CARiSMA plug-ins and Eclipse Mars with Papyrus plug-ins into a bundle wrapped together with the automating program to make a complete tool. With this bundle, the system becomes compact and can be copied easily and installable across any platform. Figures 1a, b and 2 show the different interfaces.

4 Implementation and Results

This section discusses the steps utilized in implementing and testing of the proposed system. For effective implementation of the proposed system, some design consideration must be met, such as:

i. The computer on which the system must be run must have Eclipse Mars, Kepler applications running on them, in addition to the installation of Papyrus and CARiSMA plug-in, respectively.
ii. The procurement of an appropriate computer system with enough configurations to run this system is an absolute requirement.

The specifications of the system and all the necessary requirements needed to run the proposed system were analyzed.

This system was developed and compiled as a Java executable program which could run on any operating system platform. There was no special installation required for this implementation. The process of deployment of the application is simply by executing build command on NetBeans and then copying the “dist” folder from the project directory as the Java application.

Wherever the dist folder is copied, the program can be launched by opening the folder and running the “emedical.jar” executable program.

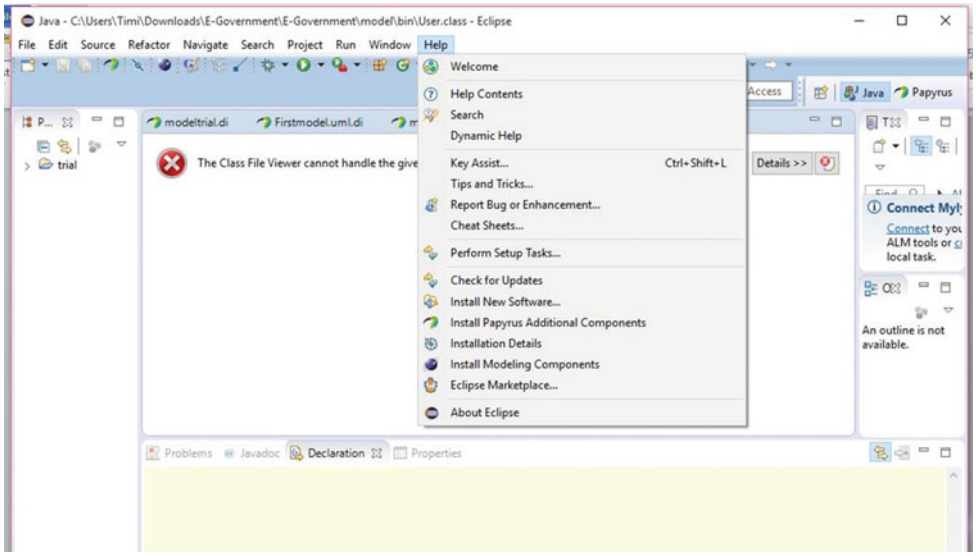

Fig. 1 Sample screen of Papyrus interface

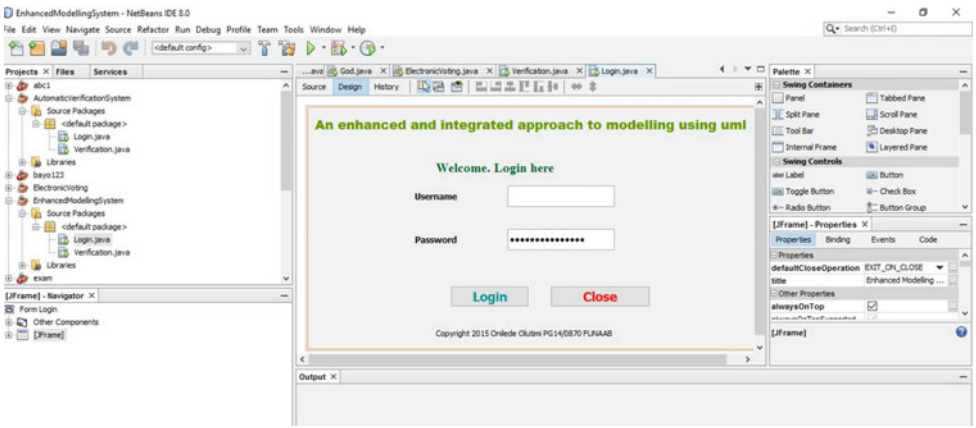

Fig. 2 Screenshot of NetBeans development environment

4.1 System Specification

This system was built with the following specifications:

i. Interactive and control-based graphical user interface.
ii. Object-Oriented Development.
iii. Java 2 Programming Platform.
iv. Native application.
v. Multi-operating system compatible program.

4.2 Results

In the system design, a login page where a user is required to login to the system with a valid username and password is presented. The tasks pane will then be displayed to the user.

The user is expected to choose a task, and the system will automatically locate the right tool and open the platform for the task.

Menu Page: This page displays the menu for the administration of the system, indicating all that can be done by a user as depicted in Fig. 3, while Fig. 4 depicts the screenshot of the modeled system.

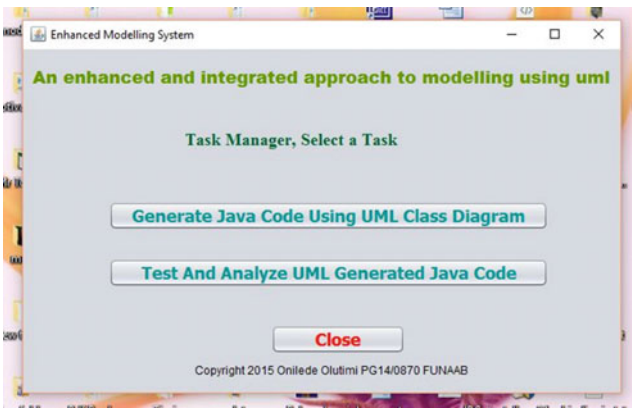

Fig. 3 Screenshot of the menu page

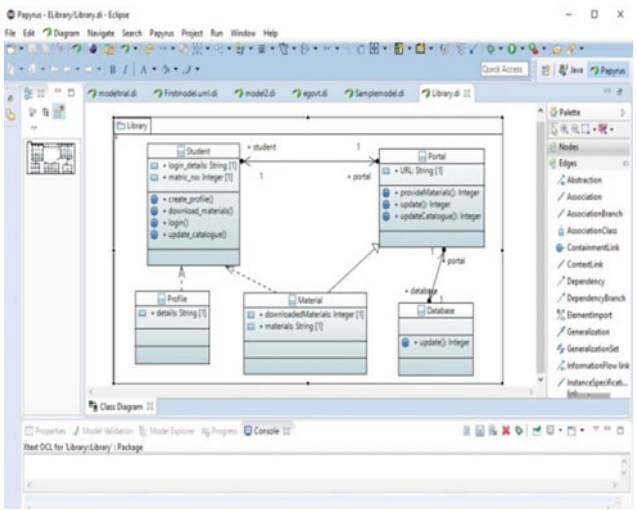

Fig. 4 Modeled system

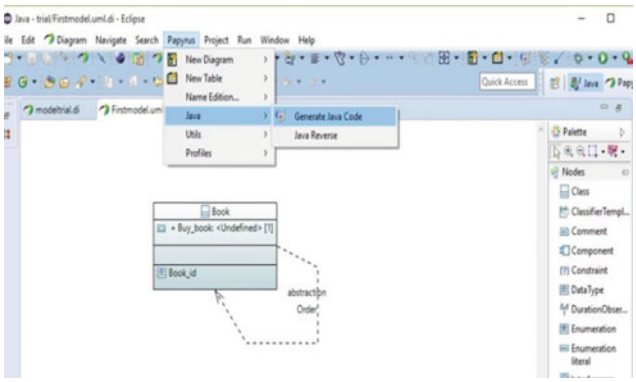

Fig. 5 Generating Java code from the modeled system

Generating Java Code: From the interface, the user can generate Java code for the diagram by going to Papyrus menu, selecting Java option and finally generating Java code as shown in Figs. 5 and 6.

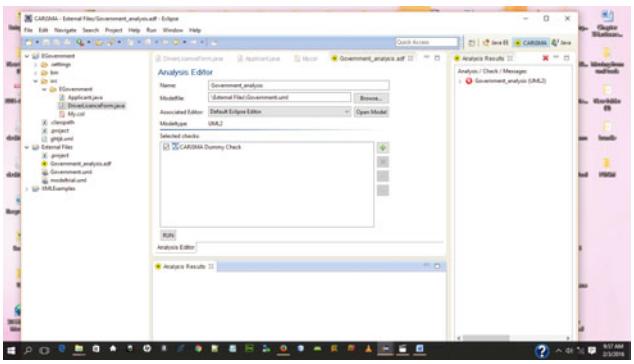

Fig. 6 Analyzing the generated code

5 Conclusion

The concept, design, construction, and implementation of this study clearly demonstrate the role of technology in driving the majority of human activities in the world today. Thus, there is no limit to the application of technology, as demonstrated in this study.

The concept and operation of this system are such that when the user intends to build an enterprise or large software, instead of going through the tedious process of writing all the code to build the software, such developer can use this tool such that he can come up with the class diagram of the system as a model and possibly build stereotypes needed to build a secure and robust system. These class diagrams can then be drawn by the Papyrus software development modeler. After drawing all the class diagrams of classes and stereotypes involved in Papyrus, it will then be converted to ODI file that will then be transported to Eclipse Kepler on which CARiSMA the analysis tool is installed.

The ODI file will be analyzed by CARiSMA, and a report on the usability and conformity of the class diagrams will be reported by the CARiSMA plug-in.

The proposed system can be adopted or implemented by any software modeler.

However, challenges identified during the implementation of the system include the difficulty and time consumption in conceptualizing the operation of the system. In addition, integrating three major Java studios on one system can also be tedious and especially while trying to configure CARiSMA and Papyrus. Finally, the development of the proposed system was quite challenging but exploratory and exposing.

Future Research Direction

The future recommendation is to develop a more robust system with a focus on UMLSecrisk. We intend to explore in-depth understanding on how to implement stereotypes with class diagram for a more enhanced system.

References

Amaral, J. N., et al. (2011). *About computing science research methodology*. Edmonton, Alberta, Canada. https://doi.org/10.1.1.124.702.

Amirat, A., Menasria, A., Oubelli, M. A., & Younsi, N. (2012). Automatic generation of PROMELA code from sequence diagram with imbricate combined fragments. In *2nd International Conference on the Innovative Computing Technology (INTECH 2012)*, pp. 111–116. IEEE.

Arogundade, O.'T., Adubiagbe, H. K., Ojokoh, B. A., & Mustapha, A. M. (2016). Design and validation of e-motoring services model. *FUTA Journal of Research in Sciences, 12*(2), 307–324.