ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Review on Code Generation from UML Diagrams

Prajkta R. Pawde[*1], Prof. V.Chole[2], Prof. P.S. Prasad[3]
prajkta1688@gmail.com

Abstract

The Unified Modeling Language (UML) [1] has now become the de-facto industry standard for object-oriented (OO) software development. UML provides a set of diagrams to model structural and behavioral aspects of an object-oriented system[2,3].Automatic code generation is efficient which, in turn, helps the software engineers deliver the software on time. This paper represents an review for generating efficient and compact executable code from UML diagram. For this we first preprocess the given diagram for its correctness and then verify if for the same and then try to optimize the obtained diagram from these stages so as to reduce redundancies present in the diagram and finally generate the code for it.
.

Keywords: UML, source, code, generation, optimization, preprocessing.

Introduction

At present and in the future, the technology development is accompanied by an increase in applications complexity. Code generators are used to increase code quality and decrease envelopment time, since their goal is to generate repetitive source code while maintaining a high consistency level of the generated program code. Code generation assumes the mission of writing repetitive code parts, leaving to programmers more time to concentrate on specific code. The generators provide more productivity; generate great volumes of code, which would take much longer if coded manually. Consistent code quality is preserved throughout the entire generated part of a project. Required coding conventions are consistently applied, unlike handwritten code, where the quality is subject to variation. In case of finding errors in generated code, the errors can be corrected in short time through revising of templates and re-running the process of code generation [4].

Nowadays, there exist many software development tools able to generate source code of basic application skeleton from its formal description unfortunately, in many cases these tools lack an ability to generate complete, production-ready source code defining both the application structure and logic suitable for building without need of any modifications done by a developer. This paper shows principles and algorithms used in cross-platform development tool called CodeDesigner RAD [5] aimed for production ready source code generation which allows users to generate complete applications from their formal description.

This paper presents an approach to generate code from UML diagrams. The most important characteristic of this approach is the preserved flexibility towards the target programming language, accomplished by code generation through two transformations; first into an intermediate code and then into the code of a selected target language. Since the complexity of UML model can vary from simple to highly complex, an approach provides an efficient way for generation code from UML diagrams.

Related Work

A series of work on code generation from UML statechart diagrams are reported by Tanaka et al. [6, 12]. Niaz and Tanaka [6] propose an approach to generate Java code from UML class and statechart diagrams. For code generation, they represent states as objects, transitions as operations, hierarchical and concurrent substates by means of object composition and delegation. To simplify the code generation process, Niaz and Tanaka introduce a helper object that encapsulates all the state specific behaviour of a multi-state domain object.

Engels et al. [12] propose a transformation process based on collaboration diagram to generate Java code. For this, Engels et al. provide the guidelines to transform the collaboration diagrams into a well-formed structure. Engels et al. consider a refined meta-model for collaborations, so that well-formed structured collaboration diagram can be

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

instantiated from the refined meta-model. The transformation rules are applied on the refined meta-model to generate Java code. HiberObjects generates code from UML 1.x SDs using templates for specific services supported by different types of objects (e.g. DAO, Persistent).

We now review the existing work [17, 5, 13] that closely resemble our work. Jakimi and El Koutbi [17] propose an approach to compose UML SDs representing a set of scenarios of a use case into a single SD. For composition of the SDs (i.e. a set of scenarios), they use four operators: sequential operator, concurrent operator, conditional operator and iteration operator. The resultant single SDs obtained for all use cases are then merged into a global single SD capturing the behaviour of the entire system. Finally, code is generated from the global single SD. The differences between our approach and Jakimi and El Koutbi approach [17] are as follows. First, our approach uses fragments to model the conditional messages in the SDs following UML 2.x syntax, whereas Jakimi and El Koutbi [17] use UML 1.x syntax to model them. Second, our approach uses the graph model (SIG) to handle method scope information. On the other hand, Jakimi and El Koutbi [17] have not reported how to handle method scope information which is necessary for the generation of code of different class methods.

Thongmak and Muenchaisri [5] propose a set of rules to transform UML SDs into Java code. For this, Thongmak and Muenchaisri map sequence and class diagrams into a meta-model and then apply the transformation rules to the meta-model to generate Java code. Thongmak and Muenchaisri [5] follow UML 1.x syntax for modelling the SDs. To improve the quality of code generation using UML SD, Usman and Nadeem [13] propose a tool approach called 'UJECTOR'. They use class diagram, SDs, and activity diagrams to generate structural as well as behavioural code. The object-oriented code structure is built from class diagram; flow of control within methods is obtained from the SDs; object manipulations are derived from activity diagrams.

Code Generation

The code generation process consists of four steps as shown in Fig. 2. First of all a source diagram is preprocessed so its structure will change in order to be more suitable for further processing by the code generator. Preprocessed diagram must be verified to find possible inconsistencies in the diagram's topology. If the verification fails the code generation process is aborted. After that, a set of optimization procedures leading to various simplifications of the diagram's structure can be performed on the verified diagram. The last step represents a final generation of a source code from verified and optimized diagram. This task is performed by a functional object called code generator.

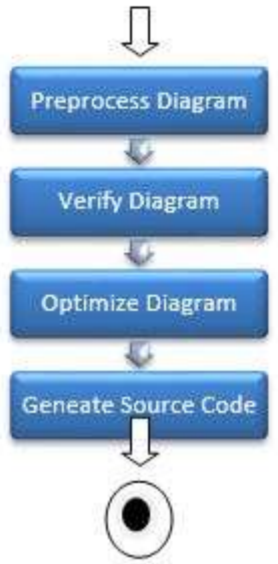

Fig. 1: Code Generation Process

The code generator reads the structure of modified diagram and writes source code fragments accordingly to the used code generation algorithm to output source file(s). Code generation algorithms can be filtered by output programming language since some language doesn't have to support all command statements produced by the algorithm. Generally, there are four basic types of code fragments:
_ functions declarations and definitions,
_ variables declarations and assignments,
_ conditional statements,
_ user-defined source code of methods/functions and the conditional statements.

Code generation algorithms use so called element processors which provide symbolic code tokens for processed diagram elements. These symbolic tokens are converted into textual code fragments by language processors with syntax in accordance to the used output programming language specification. Several language processors can be used at the same time so we can get set of source files in different programming languages during one code generation process. The complete structure of source code generator implemented in CodeDesigner RAD is shown in Fig 2

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

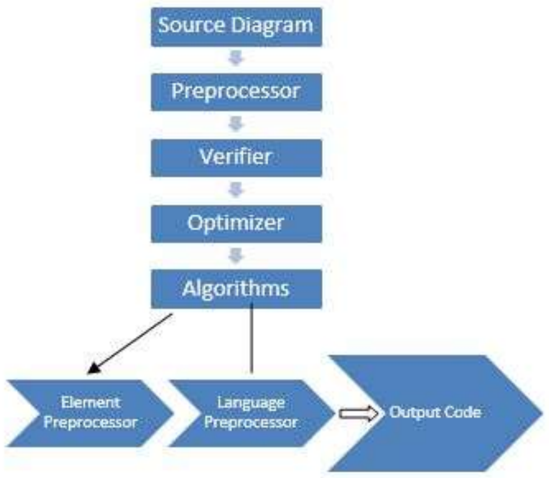

Fig. 2: Structure of code generator

A. Diagram preprocessing

Preprocessing of source diagrams implemented in CodeDesigner RAD is used for deconvolution of hierarchical state charts [5] into classic Mealy state machines [1] further processed by state chart code generator.
The deconvolution process consists of four dependent steps:

1. conversion of entry/exit state actions into transition actions
– the algorithm checks all states in the diagram whether they include entry/exit actions and assign those actions to all non-state-loop (i.e. transitions starting and ending in different states) incoming/outcoming transitions

2. Re-connection of state outputs
– the algorithm creates copies of conditional transitions starting in parent hierarchical state and connects them to all (next level) child states.The condition-less transitions starting in the parent hierarchical state will be connected to embedded final state. The child states must be processed from the top level to the bottom level so BFS algorithm [17] must be used for retrieving of the child states. The simple illustration of this modification

3. Re-connection of state inputs
– the algorithm re-connects all transitions starting in embedded initial states (i.e. initial states placed inside a hierarchical state) to their parent hierarchical state.

4. State actions sorting
– state actions assigned to transitions must be sorted in such way that EXIT actions must be at the top of the list followed by ENTRY actions.

B. Verification

Verification is done after first step for consistency checking so that all the stapes after this stage should be processed correctly.

C. Optimizing Diagram

Optimization algorithms provided by state chart generator implemented in CodeDesigner RAD are aimed to reduction of used states and to simplifications of the diagram topology. Moreover, they allow code generator to produce source code easily readable by humans. The diagram optimization process consists of three dependent steps which can be performed in several iterations until further optimization is required and a defined maximum iteration count is not reached.

D. Generator

There are many existing code generators are present for generating code from UML diagrams. As there are 14 different types of UML diagrams are present, different tools are also present for obtaining efficient code in different languages from different UML Diagrams. Many generators provide simply the skeleton for an application and then we have to write some code into it for making it executable application.

Code generation algorithms implemented in CodeDesigner RAD aggregates set of so called element processors responsible for production of source code fragments based on processed diagram element. Generally, the element processors can be shared between several code generation algorithms or several diagram elements if they produce the same source code.

Code Optimizations

Generated source code can be optimized at several different levels. Optimizations performed on processed diagrams discussed in Chapter II.C. lead to production of optimized source codes with reduced extent or better readability due to better source diagram's topology. Also, some low-level optimization can be performed to obtain efficient code. Optimization of generated helps in obtaining efficient code,eliminating unused code etc.

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

Conclusion

This paper presents an idea for code generation from uml diagram. Simple code can be implemented using this method. But complex code generation is not possible using this method.

References

[1] *Object Management Group (OMG), Unified Modeling Language Specification, Version 2.1.1, (2007-02-07).*

[2] *Booch, G., Object Oriented Design with Applications, Benjamin/Cummings, Redwood, California, 1991. ISBN: 0-8053-0091-0, ISSN: 0896-8438*

[3] *Coad, P., and E. Youdon, Object-Oriented Analysis, Prentice Hall, Eaglewood Cliffs, New Jersey, 1991. ISBN: 0-13-630070-7*

[4] *Herrington, J., Code Generation in Action, Manning, 2003.*

[5] *Michal Bližˇnák. CodeDesigner RAD homepage.* http://codedesigner.org/, 2011.

[6] *'http://www.ejb3.org/', 2 November 2010*

[7] *'http://www.altova.com/umodel/uml-code-generation.html', 2 November 2011*

[8] *'http://www.magicdraw.com/', 2 November 2010*

[9] *'http://www.visual-paradigm.com/product/vpuml/', 2 November 2010*

[10] *Niaz, I.A., Tanaka, J.: 'An object-oriented approach to generate Java code from UML statecharts', Int. J. Comput. Inf. Sci., 2005, 6, (2)*

[11] *Ali, J., Tanaka, J.: 'Converting statecharts into Java code'. Proc. Of Fourth World Conf. on Integrated Design and Process Technology (IDPT99), Dallas, Texas, USA, 200*

[12] *Niaz, I.A., Tanaka, J.: 'Mapping UML statecharts to Java code'. Proc. Of IASTED Int. Conf. on Software Engineering (SE 2004), Innsbruck, Austria, 2004, pp. 111–116*

[13] *Engels, G., Hucking, R., Sauer, S., Wagner, A.: 'UML collaboration diagrams and their transformations to Java'. Proc. of the Second Int. Conf. on the UML, 1999, (LNCS 1723) pp. 473–488*

[14] *http://objectgeneration.com/eclipse/04-sequencediagrams.html', 2 November 2010*

[15] *J. Ali, and J. Tanaka, "An Object Oriented Approach to Generate Executable Code from the OMT-based Dynamic Model", Journal of Integrated Design and Process Science (SDPT), vol. 2, no. 4, 1998, pp.65-77.*

[16] *D. Harel, and E. Gery, "Executable Object Modeling with Statecharts", 18th International Conference on Software Engineering (SE'96), IEEE Computer Society Press, Berlin, Germany, March 25-29, 1996, pp.246-257.*

[17] *Donald E. Knuth. The Art of Computer Programming Vol 1.Boston: Addison-Wesley, 3rd edition, 1997.*