Improving Requirement Specification by Prototype Generation from Requirement Models

Sunitha E V
Department of Computer Science & Engineering
Mangalam College of Engineering, Ettumanoor
Kottayam, India

***Abstract*-- Gathering, analyzing and validating the requirements is very important in software development. The completeness of the requirements plays a key role in the successful and timely delivery of the software product. Make the requirements complete through prototyping is a well-accepted approach. In this paper I propose a method to automate the prototype generation process. I propose three algorithms for the same. The algorithm 1.1 defines the service method, which implements the main features of the system. The algorithm 1.2 populates the class definitions with the method definitions. The algorithm 1.3 updates the method definitions with nested method calls. The proposed approach is implemented as a tool, and evaluated its output. It is found that the tool generates, on an average, 30% code of actual code. The proposed method of prototype generation will improve the productivity of the software industries, since it reduces the requirement gathering effort and time. Since UML is commonly used in software industry, our method is easily adaptable by software developers in industry.**

Keywords* - *Prototyping, UML, XML, Use Case, sequence diagram.

I. INTRODUCTION

The use case model is used throughout the software development. In the requirement specification phase, it is used for specifying functional requirements. This will be used in analysis and design phase as the base input. Moreover, the use case model is used as input to iteration planning, for test case generation, and as a major component for user documentation.

The readers may wonder how a description with no technical details can contribute to prototype generation. There are a few research works going on, to directly convert the software requirements to source code. Our focus is on Model Driven Development [8, 9, 10, 11, 12, 13, 20] and Executable UML [3] since these two terms are welcomed in the software industry and are widely accepted these days. Our aim is to convert the use-case model to prototype of the system. In this view, I use use-case diagrams along with sequence diagrams for prototype generation.

The use cases are summarized to a software package which satisfies the required functionalities of the system. It helps to map the system requirements with the features provided by the system. Sequence diagram alone cannot project the customer requirements satisfied by the implemented system. That's why I included use case diagram for prototype generation.

The rest of the paper is organized as follows. Section II gives the details of requirements modeling with use case and sequence diagrams. Section III explains prototype generation process and the algorithms. Section IV gives the details of related works. Section V concludes the paper.

II. REQUIREMENTS MODELING WITH USE CASE AND SEQUENCE DIAGRAMS

A use case diagram is the graphical representation of the functional requirements of a proposed system [2]. It gives the scope of the proposed system. It is mainly used for communicating with the end-users of the system. So it is always kept non technical.

The two components in a use case diagram are the use cases and the actors. A use case is a single unit of meaningful work. That means, one aspect of the behavior of a system is represented by a use case. A use case can be described with diagrams or textual descriptions. A description includes the requirements, constraints, scenarios and scenario diagrams. UML sequence diagram is used as the scenario diagram. The users of the system are marked as actors in use case diagram. An actor in a use case diagram can be something with a behavior or role. For example, a person, another system, organization etc. Actors interact with the system. The use case diagram describes how actors related to use cases. The use case diagram helps to package the user specific methods in the actor classes during prototype generation.

Sequence diagram represents the interaction between different objects and actors in a system to accomplish a functional requirement of the system [2]. The basic components in the sequence diagram are objects, actors, lifeline & activation, and messages. Objects and actors are the instances of classes. The existence of an object is represented by lifeline (the dashed vertical lines). The rectangular box on the life line shows the activation period of the object. Messages are represented with arrows from the life line of one object to another. It indicates the communication between objects.

III. PROTOTYPE GENERATION

The static structure of a system is presented using UML use case diagram and class diagram. The dynamic and behavioral aspects of the system are presented using UML sequence diagrams. Sequence diagram emphasizes the time ordering of message between objects.

For prototype generation, I first analyze the system requirements and prepare use case diagram with necessary scenario descriptions. Then develop the class diagram to show the system architecture. Further, the behavioral details of the system are designed using sequence diagram. Finally using the

information in all the three diagrams generates prototype out of it [5,6].

A. CODE GENERATION FROM USE CASE

The use case diagram is used in prototype generation to identify different services provided by the proposed system. Each service is mentioned as a use case in the diagram. Each use case will be associated with one or more actors in the system. The use cases are converted to service methods in the associated actor classes.

As the first step identify an actor in the use case diagram. Then check whether the class diagram contains the class of the actor. If not create the class. The actor may associate with multiple use cases. Each use case is added to the actor class as a service method. After adding all use cases search for another actor in the use case diagram which is not mapped to the class diagram. Repeat this procedure till I map all actors to the class diagram.

B. CODE GENERATION FROM SEQUENCE DIAGRAMS

A sequence diagram can be formally defined as follows [4, 14].

$$SD = (\sigma, m) \text{ where}$$
$$\sigma = \{x \mid x \text{ is an object / actor}\}$$
$$m = \{msg \mid msg \text{ is a message}\}$$
$$\text{where msg is a tuple.}$$
$$msg = (ob_i : C_i, ob_j : C_j, action, order)$$

The sequence diagram is a tuple which includes set of objects and set of messages send between the objects. A message, msg, contains the sender object obi of the class Ci, the receiver object obj of the class Cj, the method call action, and finally the sequence order of the message. These details are essential for code generation from sequence diagrams. The same notations have been used in the code generation algorithms proposed in this paper.

The sequence diagram is converted to the hierarchical structure tree [21, 22] before code generation. Structure tree is nothing but a tree structure of the messages passed between the objects. The nodes in the tree are the objects which are participated in the communication, and the edges are actually the messages passed between the objects. The root of the tree will be one of the actors in the use case diagram. A sample sequence diagram and its hierarchical structure tree are shown in Figures 1 and 2.

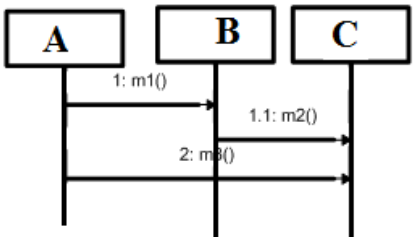

FIGURE 1. Sample sequence diagram.

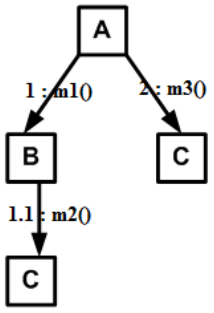

FIGURE 2. Hierarchical structure tree.

The code generation from sequence diagram has two passes. In the first pass, identify the messages and populate the classes by adding the corresponding method in the receiver class. In the second pass, the service methods will be defined.

Algorithm 1.1 Generate Services

The algorithm generate_services ( ) takes HST and the prototype P as input and returns the updated P. Each HST represents a scenario of the use case. This scenario name will be taken as the service name. Service is the method which implements a scenario of a use case. The service methods will be added to the context class. Each service method is defined as a series of method calls from the actor object. The sequence of the method calls are decided by the order of the message, msg.order. The method calls are added as objName.fnName(). Here objName I get from msg.obj and fnName from msg.action. Repeat this for all messages going from the root. Then return the updated prototype P.

Algorithm 1.2 Populate classes

The algorithm populate_classes( ) is used to add the method declaration to the classes based on the messages passed between objects of each class. The algorithm takes hierarchical structure tree (HST) of the sequence diagram and the class diagram CD as input. The output of the algorithm is the modified class diagram.

First of all, the HST is parsed to retrieve the data in the tree. Go to the root node which is normally an actor in the use case diagram. Take it as the current node, cur_node. From this point I start a breadth first traversal in the HST. Take each message edge of the cur_node and find the msg.action. This represents a method call. Check whether this method is already declared or defined in the receiver class, msg.Cj. If not, add a method declaration for msg.action in that class. Similarly take each node and find all message edges starting from it and checks whether those have corresponding method declarations in the respective receiver classes. Qpc is the queue used for executing the tree traversal. After checking each message, the receiver object obj is enqueued in Qpc. Each time, an object is dequeued from Qpc and find out the messages associated with it. Enqueue all receiver objects of those messages to Qpc. Continue this till the queue is empty. Then return the updated CD.

Algorithm 1.3 populate methods

This algorithm takes HST and CD as input. It returns the class diagram with updated method definitions. The parsed

hierarchical structure tree is used for updating method definitions. First the toot node is taken as the current node, cur_node. There can be many messages initiating from cur_node. These messages will be taken one by one. Already considered messages are labeled as 'read'. If there exists an unread message, then open the receiver class (msg.Cj) of the message and the method definition that is being called (msg.action). then push the cur_node to stack for future reference. Set new cur_node as receiver node (msg.Cj). Find all messages initiated from the cur_node and append the corresponding method call statements to the method definition. Repeat the whole process until there are no more messages to read. Then, backtrack to the parent node by popping it out from the stack and set as cur_node. Finally return the updated class diagram.

IV. RELATED WORKS

Sengupta et al. [26] depicts how the behavioral models can be converted to XML and to other behavioral diagrams. They give a method to convert sequence diagram to state machine. Their focus is on the model transformation.

Q Long [27] illustrates an algorithm to convert sequence diagram and class diagram to a target language, rCOS (Relational Calculus of Object Systems). It will first check the consistency of the class diagram and sequence diagram. It generates an error report if the diagrams are not consistent, otherwise the diagrams will be given for code generation.

Harrison [28] depicts a mapping method that converts the abstract system models to a high-level skeletal implementation code. UML is used as the design language and Java as the implementation (target) language. Harrison [28] considers class diagram for code generation. Classes marked with the stereotype <<entity>> will be mapped into an interface and a pair of implementing classes. One class will be abstract class and the other one will be instantiable. They also present the problems of generating object-oriented language implementation code from high-level designs. They summarize the issues for Java implementation, like how to handle multiple inheritance etc.

Bajwa [29] presents a rule-based production system for code generation in java. They take the requirement scenario in English language and will automatically generate UML diagrams for these scenarios. It has mainly five steps. In step 1 they accept the text input, i.e., the scenario description in English. In step 2 they do the text understanding using natural language processing. The knowledge extraction will be done in step 3. In this step, classes and objects, and their attributes will be identified. In step 4, UML class diagrams will be drawn based on the knowledge extracted in step 3. Finally, the skeleton code is generated, based on classes, in step 5.

V. CONCLUSION

In this paper I propose a method to generate prototype of a software system from the UML use-case diagram and sequence diagram. Use-case diagram is used to frame the context class and the sequence diagram is used to add details to the class. The code generation from use case diagram is done in five steps and from sequence diagram is done in three steps. Algorithm for each one is given in the paper. The algorithms give a formal way to do the prototype generation and this method is easy to implement. The analysis of the proposed method shows that it can generate even more than 30% of code for frequently interacted classes. This is a promising result.

REFERENCES

[1] Gomaa, Hassan, and Erika Mir Olimpiew. "The role of use cases in requirements and analysis modeling." *Proc. of the 2nd Intern. Workshop on Use Case Modeling (WUsCaM-05): Use Cases in Model-Driven Software Engineering*. 2005.

[2] Jacobson, Ivar. *Object-oriented software engineering: a use case driven approach*. Pearson Education India, 1993.

[3] Mellor, J. Stephen, Balcer, M., and Foreword By-Jacoboson, I.," Executable UML: A foundation for model-driven architectures", Addison-Wesley Longman Publishing Co., Inc.., 2002

[4] Li, Xiaoshan, Zhiming Liu, and He Jifeng. "A formal semantics of UML sequence diagram." Software Engineering Conference, 2004. Proceedings. 2004 Australian. IEEE, 2004.

[5] Campos, Ruben. "Model Based Programming: Executable UML With Sequence Diagrams." *A Thesis Presented to The Faculty of the Computer Science Department, California State University*. (2007):

[6] Hitz, Martin, and Gerti Kappel. "Developing with UML–Some pitfalls and workarounds." International Conference on the Unified Modeling Language. Springer Berlin Heidelberg, 1998.

[7] France, Robert B., et al. "Model-driven development using UML 2.0: promises and pitfalls." *Computer* (Volume: 39, Issue: 2, Feb. 2006)): 59-66.

[8] Beydeda, Sami, and Matthias Book. *Model-driven software development*. Ed. Volker Gruhn. Vol. 15. Heidelberg: Springer, 2005.

[9] T Stahl, M Voelter, K Czarnecki, "Model-Driven Software Development: Technology, Engineering, Management", John Wiley & Sons, 2006 , ISBN:0470025700

[10] Alvarez, María Luz, et al. "A Methodological Approach to Model-Driven Design and Development of Automation Systems." IEEE Transactions on Automation Science and Engineering (2016).

[11] Lano, Kevin, and Shekoufeh Kolahdouz-Rahimi. "Model-transformation design patterns." IEEE Transactions on Software Engineering 40.12 (2014): 1224-1259.

[12] Milicev, Dragan. "Automatic model transformations using extended UML object diagrams in modeling environments." IEEE Transactions on Software Engineering 28.4 (2002): 413-431.

[13] Sendall, Shane, and Wojtek Kozaczynski. Model transformation the heart and soul of model-driven software development. No. LGL-REPORT-2003-007. 2003.

[14] Sunitha E.V, Philip Samuel, "Automatic code generation using unified modeling language activity and sequence models", IET Software, ISSN 1751-8806, Volume 10, Issue 6, 2016, pp. 1–9

[15] Jakimi, A., El Koutbi, M.: 'An object-oriented approach to UML scenarios engineering and code generation', Int. J. Comput. Theory Eng., 2009, 1, (1), pp. 35–41.

[16] Thongmak, M., Muenchaisri, P.: 'Design of rules for transforming UML sequence diagrams into Java code'. Proc. of the Ninth Asia-Pacific Software Engineering Conf. (APSEC, 2002), IEEE Computer Society, Washington, DC, 2002, pp. 485–494

[17] Parada, A.G., Siegert, E., de Brisolara, L.B.: 'Generating Java code from UML class and sequence diagrams'. Proc. of the 2011 Brazilian Symp. on Computing System Engineering, 2011, pp. 99–101

[18] EL B. Omar, B. Brahim and G. Taoufiq, "Automatic code generation by model transformation from sequence diagram of systems internal behavior", *Int. J. of Computer and Inf. Tech.*,vol. 1, no. 2, 2012.

[19] D. Kundu, D. Samanta, and R. Mall , "Automatic code generation from unified modelling language sequence diagrams", *IET Software* , vol.7 , no. 1, pp. 12-28, 2013.

[20] Solomencevs, Artūrs. "Comparing Transformation Possibilities of Topological Functioning Model and BPMN in the Context of Model Driven Architecture." *Applied Computer Systems* 19.1 (2016): 15-24.

[21] Chitra, M. T., and Elizabeth Sherly. "Refactoring sequence diagrams for code generation in UML models." *Advances in Computing,*

Communications and Informatics (ICACCI, 2014 International Conference on. IEEE, 2014.

[22] Philip Samuel, Rajib Mall, Pratyush Kanth, "Automatic test case generation from UML communication diagrams", Elsevier, Science Direct, Journal of Information and Software Technology, Vol 49, 2007 pp. 158–171.

[23] Śmiałek, Michał, Norbert Jarzębowski, and Wiktor Nowakowski. "Translation of use case scenarios to Java code." *Computer Science* 13 (2012): 35-52.

[24] N Debnath, A Funes et. al. "Integrating OCL Expressions into RSL Specifications", IEEE EIT 2007 Proceedings, 2007.

[25] J Cabot et. al. "Verification of UML/OCL Class Diagrams using Constraint Programming", Proceedings of the IEEE International Conference on Software Testing Verification and Validation Workshop, ICSTW'08, 2008.

[26] S Sengupta et. al. "Automated Translation of behavioral models using OCL and XML", TENCON 2005, IEEE Region 10, 2005

[27] Q.Long, Z.Liu et.al., "Consistent Code Generation from UML Models", Proceedings of Australian Software Engineering Conference, 2005.

[28] William Harrison, Charles Barton, Mukund Raghavachari, "Mapping UML designs to Java", Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pp: 178 - 187 , 2000.

[29] Imran Sarwar Bajwa, M. Imran Siddique, M. Abbas Choudhary, "Rule based Production Systems for Automatic Code Generation in Java", Digital Information Management, 1st International Conference, 2006. Page(s):300 – 305.

[30] Ruben Campos, "Model Based Programming: Executable UML with Sequence Diagrams", CS Thesis, California State University, Los Angeles, 2007

[31] X Li, Z Liu, "Prototyping System Requirements Model", Journal Electronic Notes in Theoretical Computer Science (ENTCS), Volume 207, April, 2008

[32] L Yin et. al. , "Modeling and Prototyping Business Processes in AutoPA", Fifth International Symposium on Theoretical Aspects of Software Engineering (TASE), 2011

[33] J Koehler et. al. "Declarative techniques for model-driven business process integration", IBM Systems Journal, Vol 44, issue 1, 2005

[34] Paetsch, Frauke, Armin Eberlein, and Frank Maurer. "Requirements engineering and agile software development." Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on. IEEE, 2003.

[35] Kamalrudin, Massila, John Hosking, and John Grundy. "Improving requirements quality using essential use case interaction patterns." Proceedings of the 33rd International Conference on Software Engineering. ACM, 2011.