XMI2UC: An Automatic Tool to Extract Use Cases from Object Oriented Source Code

Shouki A. Ebad and Moataz A. Ahmed

***Abstract*—In object oriented software engineering, use cases (UCs) describe the services offered to the users of a system by the system itself. Each UC is realized by one or more sequence diagrams (SDs) that depict how the objects interact and work together to provide the corresponding service. There are several tools that can be used to reverse engineer source code to provide SDs, e.g., Enterprise Architect, Together, and Altova UModel. However, SDs generated from such tools represent detailed "run time behavior" rather than analysis and design level models. In this paper, we present a tool, XMI2UC, that extracts the UCs from object oriented source code. We demonstrate the use of the XMI2UC tool using two case studies: a small-sized hypothetical system and a medium-sized open source jHotDraw system. The experiments show how our new tool simplified the task of extracting the UCs for object oriented systems.**

***Index Terms*—Object oriented software engineering, reverse engineering, UML, use case, sequence diagram.**

I. Introduction

The UML (Unified Modeling Language) [1] is widely used to express early-stage Object Oriented artifacts (e.g., analysis and design models). It is a language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML representations of these artifacts are referred to as UML models (diagrams). UML taxonomy of diagrams provides a logical organization for the various major kinds of diagrams into only two major categories: structure and behavior; with no category to represent the functional aspect. It is worth noting here through that there is a general paucity of concepts for specifying the functionality of object communities. Nevertheless, use cases (UCs) can be interpreted as one means of specifying functionality according to Jacobson [2][1]. UCs define the functionality of the system and constitute a specific way of modelling some part of this functionality. Clearly, a UC has also a flavor of behavior abstraction, as it is a special sequence of related transactions in the interaction between the actor and the system. Accordingly, the functionality (i.e., the services) that users require of the OO system is documented in use cases. Use cases describe the typical interactions between the users of a system and the system itself, providing a narrative of how a system is used. Each use case is realized by one or more sequence diagrams (SDs) that depict how the objects interact and work together to provide services. Each individual object provides only a small element of the functionality – its particular responsibilities – but when they work together, objects are able to produce services that people can use. There are several tools that generate SDs at source code level such as Enterprise Architect[1], Together[2], and Altova UModel[3]. Usually, SDs generated from source code are "run time behaviour" since they are based on methods behaviour at run time. In other words, when you specify a particular method, the run time system can generate the behaviour of this method across the entire system. Up to author's knowledge, there is no tool that generates the SDs which reflect the functionality of the system i.e., UCs. Therefore, we have developed a tool called XMI2UC (from XMI[4] to Use Cases) with less than 250 Java LOC to extract the UCs from source code (reverse engineering) of object oriented software systems via XMI documents. The structure of this paper is as follows: Section 2 presents modelling of XMI2UC process using DFD. Section 3 describes a hypothetical case study for a small-sized system. Another real case study for a medium-sized open source jHotDraw project is described in Section 4. Section 5summarizes the contributions of our work and outlines our future plans to improve XMI2UC.

II. XMI2UC Modeling

A data flow diagram (DFD), modeling XMI2UC process, is presented in Fig. 1. Using Altova UModel application, the source code is imported to generate the needed SDs. This type of SD is run-time SDs (i.e., does not reflect the system functionality. Because of this, a transient task should be taken place after generating the SDs; it is a filtering process. Filtering is a manual means to eliminate all run-time variables and messages shown in UML sequence diagrams generated by Altova.

Real software projects usually contain a lot of statements representing the run time environment such as loop and control statements. Filtering process makes the generated "run-time" SDs mimic the "functional" SDs that describe the "functional" behaviour of UCs through objects and messages starting from the pre-condition to the post-condition. After filtering SDs, the whole system exported in XMI document to be used as input to the XMI2UC tool. XMI2UC now looks for specific "patterns" or "XMI elements". Patterns identified include: packages, classes,

Manuscript received April 10, 2012; revised May 5, 2012

The authors are with Information and Computer Science Department King Fahd University of Petroleum and Minerals Saudi Arabia, Dhahran 31261.

[1] http://www.sparxsystems.com.au/
[2] http://www.borland.com/us/products/together/
[3] http://www.altova.com/umodel.html
[4] XMI is abbreviation to XML Metadata Interchange

methods, interactions, call events, and sequence diagrams to produce three outcomes: (1) Package Information, (2) Class Information, and (3) Use Case Information.

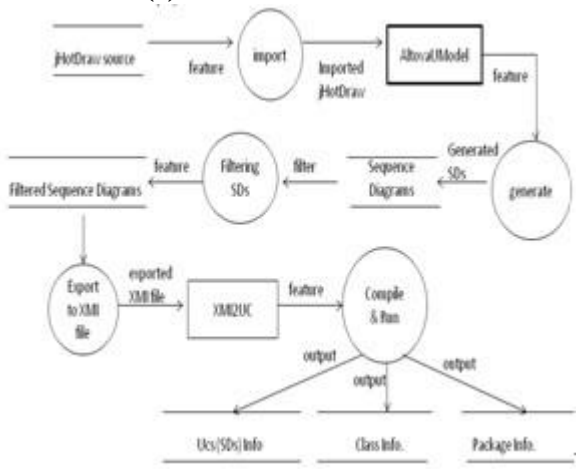

Fig. 1. DFD of the XMI2UC process.

III. A Hypothetical Case Study

Fig 2 presents an example to outcomes produced by XMI2UC tool. It is shown the system consists of seven classes distributed over two packages: P_1 and P_2 where P_1 has three classes: A with two methods: am_1 and am_2, B with two methods: bm_1 and bm_2, C with three methods: cm_1, cm_2, and cm_3. The second package P_2 consists of four classes: D with method dm_1, E with method em_1, F with two methods: fm_1 and fm_2, and finally G with method gm_1. The most important outcome, UCs Information, shows the system has three UCs: UC_1, UC_2, and UC_3; each realized by SD as appeared in Fig. 3. The 'UCs Information' product describes the path of each UC; i.e., the sequence of related transactions (methods).

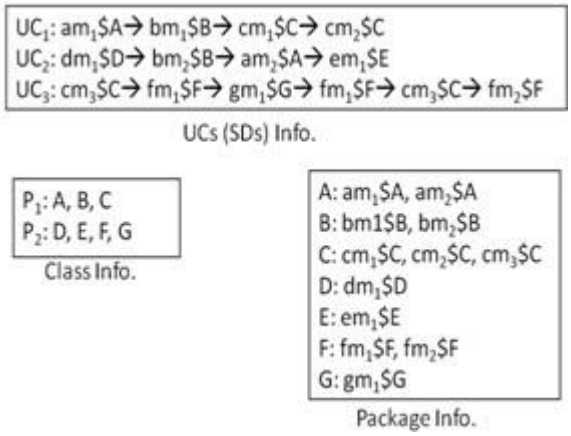

Fig. 2. The XMI2UC output products of the hypothetical system.

As it is seen with the above hypothetical system, the methods are named semantically so that the naming style indicates to two things: class that the method belongs to and the order of the method within the total methods of the class. For example, the name (bm_2) indicates to a method (m) in class B; and because of the subscription number 2, there is at least one more method existing in B which is bm_1. In real case systems, however, there may be several methods with the same name located in different classes such as getter and setter methods. In such a case, XMI2UC tool made a small modification to the method's naming convention to fix such a problem so that the class name is concatenated to the method name using '$' symbol. In this case, no side effect with different methods having the same name located at different classes. Even though the method overloading concept looks a bit confused with this naming convention, the matter is different because calling such methods is done dynamically (i.e., based on some run-time variables such as the data type of method arguments) which should be removed during the filtering process.

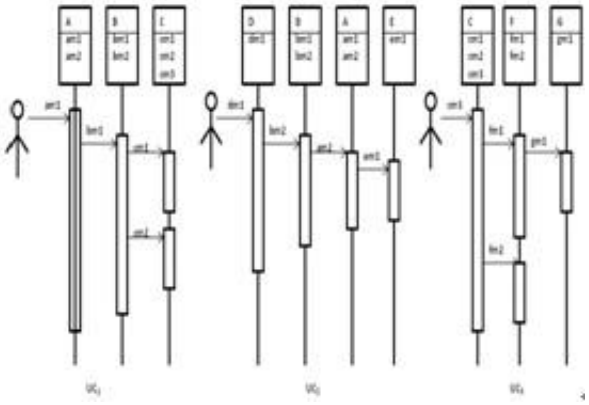

Fig. 3. UCs (realized by SDs) of the hypothetical system.

Overlapping among the UCs (i.e., SDs generated by Altova UModel) is another issue. It reflects existence of redundancy in the considered UCs without any additional value. Back to the hypothetical system consisting the three UCs (realized by three SDs) shown in Fig 3; assume there are two more UCs existing in the system. Their paths are as follows:

UC_4: $bm_1 \rightarrow cm_1 \rightarrow bm_1 \rightarrow cm_2$

UC_5: main $\rightarrow dm_1 \rightarrow bm_2 \rightarrow am_2 \rightarrow em_1$

When the XMI2UC tool takes all the five UCs, two interactions, $bm_1 \rightarrow cm_1$ and $bm_1 \rightarrow cm_2$, would be repeated because they belong to two different UCs: UC_1 and UC_4. The same thing would be happen with UC2 and UC5 where three interactions would be repeated too: from $dm_1 \rightarrow bm_2$, $bm_2 \rightarrow am_2$, and $am_2 \rightarrow em_1$.

The XMI2UC tool remedies this problem by making a decision to adapting the suitable UC that reflects functionality without redundancy. This decision is made dynamically i.e., during the extraction UCs from XMI document. In particular, XMI2UC discovers the wide UC to adapt it and the narrow UC to remove it. So the developer should be aware that it is not necessarily all UCs (realized by SDs) generated by Altova UModel are considered in XMI2UC tool. Generally, the relation between the UCs in both AltovaUModel and XMI2UC could be described as flows:

$$number\ of\ UCs\ generated\ by\ XMI2UC \leq number\ of\ UCs\ generated\ by\ Altova$$

As a result, despite there are five UCs in the hypothetical system, XMI2UC tool will produce only three: UC1, UC3, and UC5 while the other two are removed even though they are generated by Altova UModel. The above mathematical relation is satisfied here where number of UCs processed in

XMI2UC is 3 while those generated by Altoval UModel is 5; and $3 \leq 5$.

IV. jHotDraw Case Study

jHotDraw (www.jhotdraw.org/) is a Java GUI framework that can be used for developing custom-made drawing editor applications. This application was used as a case study by some researchers such as Seng et al. 2006 [3]. We applied the proposed XMI2UC tool to jHotDraw version 5.3. Despite this version consists of nine packages, we concentrated on the four: *contrib*, *figures*, *standard* and *util* that make up most of the application's core. The number of UCs (i.e., SDs) generated by Altova was ~180 while the number of UCs generated by XMI2UC was 165; i.e., more than 50% of the UCs are removed because of existence of UCs overlapping. Because of paper limit, we present a sample of the 'UCs Information' produced by XMI2UC. Table 1 shows 15 UCs generated by XMI2UC for the jHotDraw5.3.

TABLE I: Sample of UCs Generated by XMI2UC for The..

UC#	UC Path
UC_1	chop$ChopDiamondConnector→pointToAngle$Geom→chop$ChopDiamondConnector →angleToPoint$Geom
UC_2	handlePopupMenu$CustomSelectionTool→setSelectedFigure$PopupMenuFigureSelection
UC_3	mouseDown$CustomSelectionTool→setSelectedFigure$PopupMenuFigureSelection
UC_4	mouseUp$CustomSelectionTool→setSelectedFigure$PopupMenuFigureSelection
UC_5	showPopupMenu$CustomSelectionTool→setSelectedFigure$PopupMenuFigureSelection
UC_6	chop$DiamondFigure→chop$PolygonFigure→length2$Geom
UC_7	add$MDIDesktopPane→resizeDesktop$MDIDesktopManager→setAllSize$MDIDesktopPane
UC_8	basicDisplayBox$GraphicalCompositeFigure→layout$Layouter
UC_9	layout$GraphicalCompositeFigure→calculateLayout$Layouter →layout$GraphicalCompositeFigure→layout$Layouter
UC_{10}	update$GraphicalCompositeFigure→calculateLayout$Layouter →update$GraphicalCompositeFigure→layout$Layouter
UC_{11}	basicDisplayBox$PolygonFigure→length$Geom
UC_{12}	chop$TriangleFigure→chop$PolygonFigure→length2$Geom
UC_{13}	findSegment$PolyLineFigure→lineContainsPoint$Geom
UC_{14}	joinSegments$PolyLineFigure→length$Geom
UC_{15}	invokeEnd$UndoActivity→getArc$RoundRectangleFigure→invokeEnd$UndoActivity →getOldRadius$UndoActivity

jHotDraw5.3 case study

V. Conclusion and Future Work

The main contribution of this paper is developing an automatic tool that extracts the UCs from source code. Two case studies are presented to demonstrate the use of XMI2UC. We plan to conduct more case studies to demonstrate the effectiveness and any limitations we find in this version. So far, our case studies have involved Java programs, we plan to study C++ programs.

Acknowledgements.

The authors wish to acknowledge KFUPM for supporting this research.

References

[1] Object Management Group, Unified Modeling Language UML, v2.3, May 2010, [Online]. Available: http://www.uml.org.

[2] I. Jacobson, Object-oriented software engineering: a use case driven approach, Addison-Wesley, 1992.

[3] O. Seng, M. Bauer, M. Biehl, and G. Pach, "Search Search Based Determination of Refactorings for Improving the Class Structure of Object Oriented Systems," in *Proc. of the GECCO'06*, July 8-12, 2006.