An Analysis and New Methodology for Reverse Engineering of UML Behavioral

Chafik Baidada, El Mahi Bouziane, Abdeslam Jakimi

Software Engineering & Engineering of Information Systems Group, Computer Sciences Department, Moulay Ismaïl University, FST Errachidia, Morocco

Abstract— *The emergence of Unified Modeling Language (UML) as a standard for modeling systems has encouraged the use of automated software tools that facilitate the development process from analysis through coding. Reverse Engineering has become a viable method to measure an existing system and reconstruct the necessary model from its original. The Reverse Engineering of behavioral models consists in extracting high-level models that help understand the behavior of existing software systems. In this paper we present an ongoing work on extracting UML diagrams from object-oriented programming languages. we propose an approach for the reverse engineering of UML behavior from the analysis of execution traces produced dynamically by an object-oriented application using formal and semi-formal techniques for modeling the dynamic behavior of a system. Our methods show that this approach can produce UML behavioral diagrams in reasonable time and suggest that these diagrams are helpful in understanding the behavior of the underlying application.*

Keywords— Reverse Engineering, UML, Behavior diagrams; Execution traces.

I. INTRODUCTION

Recently, software is becoming increasingly complex. Traditional software engineering research and development focuses on increasing the productivity and quality of systems under development or being planned. Without diminishing the importance of software engineering activities focusing on initial design and development, empirical evidence suggests that significant resources are devoted to reversing the effects of poorly designed or neglected software systems. In a perfect world, all software systems, past and present, would be developed and maintained with the benefit of well-structured software engineering guidelines. In the reality, many systems are not or have had their structured design negated. The understanding of these programs is an essential part of maintenance, reuse, validation and other activities of software engineering. An important part of maintenance time is often devoted to reading the code to understand the functionality of the program. According to some studies, up to 60% of the maintenance is devoted to understanding the software [1].

Therefore, it is important to develop tools and techniques that facilitate the task of understanding such systems because the documentation is often absent, outdated or incomplete. An effective recognition technique to understand such programs is reverse engineering. In the world of object-oriented, target language most used for reverse engineering is UML [2] due to its significant presence in the industry.

The remainder of this paper is organized as follows. In the next section, we present a general idea of the UML and reverse engineering relevant to this work. Section 3 provides background about various approaches to implement UML behavior. In section 4, we explain our research method. Section 5 gives result and discussion of approach. Finally, in section 6, we summarize and conclude.

II. UML AND REVERSE ENGINEERING

A. Objectives of Reverse Engineering

The primary purpose of reverse engineering a software system is to increase the overall comprehensibility of the system for both maintenance and new development. When we try to characterize reverse engineering in terms of its objectives, there are six key objectives [3].

1. ***Cope with complexity:*** We must develop method to better deal with the share volume and complexity of the system. A key to controlling these attributes is automated support. Reverse Engineering methods and tools, combined with case environments, will provide a way to extract relevant information so decision makers can control the process and the product in the system.
2. ***Generate alternate views***: Graphical representation has long accepted as comprehension aids. However, creating and maintaining them continuous to be a bottleneck in the process. Reverse-engineering tools facilitate the generation or regeneration of graphical representation from other forms. While many designers work from a

single, primary perspective (like dataflow diagrams), reverse-engineering tools can generate additional views from other perspective (like control flow diagram, structure chart, and entity relationship diagrams) to aid the review and verification process.

3. ***Recover lost information***: The continuing evolution of large, long –lived systems leads to lost information about the system design. Modifications are frequently not reflected in the documentation, particularly at a higher level than the code itself. While it is no substitute for preserving design history in the first place, reverse-engineering – particularly design recovery is our way to salvage whatever we can form the existing system.
4. ***Detect side effect***: Both haphazard initial design and successive modification can lead to unintended ramification and side effects that impede a system‟s performance in subtle ways.
5. ***Synthesize higher abstraction***: Reverse-engineering requires methods and techniques for creating alternate views that transcend to higher abstraction levels. There is a debate in the software community as to how completely process can be automated. Clearly, expert-system technology will play a major role in achieving the full potential of generating high-level abstraction.
6. ***Facilitate reuse***: A significant issue in the movement toward software reusability is the large body of existing software assets. Reverse- engineering can help detect candidates for reusable software components from present system

B. UML Behavioral

UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. UML includes a set of graphic notation techniques to create visual models of object-oriented software systems. UML combines techniques from data modeling, business modeling, object modeling, and component modeling and can be used throughout the software development life-cycle and across different implementation technologies. UML diagrams represent two different views of a system model:

1. Static (or structural) view: This view emphasizes the static structure of the system using objects, attributes, operations, and relationships. Ex: Class diagram, Composite Structure diagram.
2. Dynamic (or behavioral) view: This view emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. Ex: Sequence diagram, Activity diagram, State Machine diagram.

UML 2.2 has 14 types of diagrams divided into two categories:

1. *Structure Diagrams*: These diagrams emphasize the things that must be present in the system being modeled. Since they represent the structure, they are used extensively in documenting the software architecture of software systems.
2. *Behavior Diagrams*: These diagrams emphasize what must happen in the system being modeled. Since they illustrate the behavior of a system, they are used extensively to describe the functionality of software systems.

Object oriented analysis and design methods offer a good framework for behavior. In this work, we adopted the UML, which is a unified notation for object oriented analysis and design.

UML is a good target language for the reverse engineering of models since it is largely used and offers different diagrams. The reverse-engineering of UML static diagrams – like class diagrams – has been studied and is now implemented in several tools. Static views of the system allow engineers to understand its structure but it does not show the behavior of the software. To fully understand its behavior, dynamic models are needed, such as sequence diagrams or statecharts.

C. Reverse Engineering of UML

Reverse engineering is the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at higher levels of abstraction [1]. Reverse engineering is needed when the process to understand a software system would take a long time due to incorrect, out of date documentation, complexity of the system and the insufficient knowledge of the maintainer of the system. Reverse Engineering is used to recover lost information, to improve or provide documentation, to detect side effects, to reuse the components and to reduce the maintenance effort.

Nowadays, UML is the most adopted modeling language for system design in the software engineering organizations and industry. The main reasons are: (i) its friendly and intuitive notations, (ii) availability of commercial and open source tools that support the UML notations and (iii) autonomy of particular programming languages and development processes. This technique is widely used in several disciplines including new technologies such as mechanics, electronics, etc. In computer science, Reverse engineering dynamic models is to convert the code to models such as UML sequence diagram, state diagram, etc, the goal is to increase the level of abstraction to better understand the behavior of software.

Reverse engineering dynamic UML models is a very efficient way to understand complex software whose source code is

absent or documentation is outdated. It is also used in software engineering activities such as testing and validation.

In this paper, we consider UML diagrams as target design models for reverse-engineering. This means that we focus only on reverse-engineering of UML diagrams (i.e. the process that analyzes the execution of the system and creates UML diagrams that depicting its structure and its behavior). UML is an object-oriented language for software system modeling. It is composed of a set of diagrams which allow specifying the several aspects of a system. The two main aspects are: static (structural diagrams) and behavior (dynamic) aspects. The static aspect of system can be specified using class or component diagrams, while the behavior aspect can be specified using sequence diagrams, state machines and activity diagrams...etc.

In the earlier approaches, reverse engineering of object oriented code resulted in generation of class diagrams, interaction diagrams and use case model in general. However they represent only the static nature of the object oriented systems.

III. RELATED WORKS

Several studies have been performed on the reverse engineering of UML diagrams [4,5,6,7,8,9,10,11]. We distinguish two categories in existing approaches: static and dynamic. Static analysis is to use the code structure to generate the sequence diagram. One of the main works based on static analysis is that Rountev et al. [4]. They proposed an approach for the extraction of UML sequence diagrams from code through building the control flow graphs. The dynamic analysis is to analyze the performance of the application. Several studies try to generate the sequence diagram by analyzing the execution traces. In [5] is proposed an approach to build a high-level sequence diagram incrementally from basic diagram using the operators introduced by UML 2.0. In [6] they try to build a high-level sequence diagram from combined fragment using the state vector describing the system. In [7], it is proposed an approach completely dynamic based on the LTS (labeled transition system) for merger traces collected and generate a high-level sequence diagram.

These approaches have succeeded in generating representative UML behavior. But they recognize some limitations. These limitations include the information filtering problem. Thus, the resulting sequence diagram contains a lot of useless information that does not help to understand the software.

IV. RESEARCH METHOD

The reverse engineering of behavioral models consists in extracting high-level models that help understand the behavior of existing software systems. Our approach for reverse engineering of UML behavior diagrams is defined in four main steps (Fig 1.): (i) traces generation, (ii) traces collection and filtering, (iii) traces transformation into formal/semi-formal techniques and (iv) UML diagram extraction.

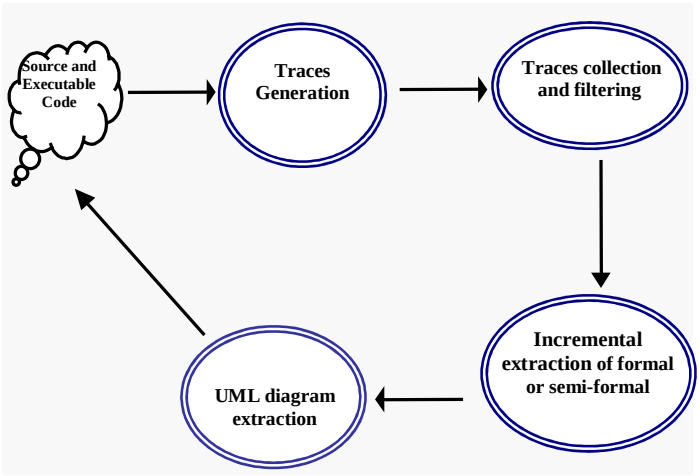

Fig.1: Overview of the proposed process

(i). ***Traces generation***. To extract high level UML behavior diagrams from an oriented-object programs, we concentrate on reverse engineering relies on dynamic analysis. As mentioned by [8, 9], dynamic analysis is more interesting suited to the reverse engineering of behavior diagram of object-oriented systems because of inheritance, polymorphism and dynamic binding. This dynamic analysis is usually performed using execution trace. There are multiple ways to generate execution traces [1]. This can include instrumentation of source code, virtual machines (ex: java programs) or the use of a customized debugger.

(ii). ***Traces collection and filtering.*** Our aim at this stage is to collect the major events occurring during the system executions. The system behavior is related to the environment entry data, in particular, values introduced by the user to initialize specific system variables. Thus, one execution session is not enough to identify all system behaviors. So we chose to run the system several times to generate different executions traces. Each execution trace corresponds to a particular scenario of a given service (use case) of the system. After that, a filtering process is applied for traces. This process is based on the package of the object present in the line trace.

(iii***). Incremental extraction of formal or semi-formal techniques***. This is the main step of our approach. It deals with the known problem of analyzing traces. Indeed, one of the

major challenges to reverse engineering high level behavior diagram is to analyzing the multiple execution traces to identify common and method invocations throughout the input traces. In this sub-section, we present our approach which uses formal or semi-formal techniques to deal with this problem. We formalized and developed an process that takes several execution traces as input and generate incrementally a formal technique that represents the system behavior.

(iv). ***UML diagram extraction.*** In this activity, we generate and build the UML diagrams behavioral using the transformation models rules (static and dynamic).

V. RESULTS/DISCUSSION

Reverse engineering is one the essential parts of software maintenance. It involves high risk especially when the maintenance and development teams are different. Extracting the static model from the source code may not be motivating factor for software maintenance. Many reverse engineering tools and approaches are proposed in literature [4,5,6,7,8,9,10,11,12,13]. However each represents only a subset of operational requirements. The process of reverse engineering is resulting in models consisting of only the static parts of design. Though the communication among objects is transformed, but reverse engineering of the internal state of the object is not presented. Static models are limited in their usefulness. It is important to realize that quality attributes such as performance and reliability can be predicted from the dynamic behavioral models of the system.

Many works investigate the reverse-engineering of UML static models, such as class diagrams. However, there is little work on reverse-engineering dynamic models, although, in addition to static models, the UML includes notations to specify the dynamic behaviour of programs, such as sequence diagrams and statecharts.

Dynamic models of programs are as important as static models because they allow maintainers to identify complex interactions among objects and to disambiguate message sends when inheritance, delegation, polymorphism, dynamic binding, re°ection are used intensively (for example, when using design patterns such as the Abstract Factory, Observer).

In this paper we proposed a new methodology to extract UML dynamic behavior diagrams from the source code (ex. java, c#) using both the static and dynamic analysis. This approach deals with the reverse engineering from execution traces for object-oriented software. Our approach uses a different methodology to deal with the problem of execution traces analysis.

There are two main categories of existing UML behavior reverse engineering approaches: the first category refers to approaches which are based on static analysis while the second concerns dynamic analysis based approaches. Static analysis is done on static information which describes the structure of the software as it is written in the source code. However, dynamic analysis is based on the system runtime behavior information which can be captured by separated tools as in [7], by instrumentation techniques as in [8], or by debugging techniques.

M. L. Nelson [14] gives a detailed discussion of the practical difficulties involved in the reverse engineering. These difficulties include that of the choice of the level of abstraction needed, and that of the formal/cognitive distinction. Computers and programming languages are formal, while the human cognitive capabilities are non- formal. Therefore, the result of any reverse engineering work could be very subjective. Any program is “understood to the extent that the reverse engineer can build up correct high level chunks from the low level details available in the program.”

We show in this paper the importance of the reverse engineered static and dynamic views of the system architecture in order to predict the system quality attributes and analyze possible plans of the system evolution.

VI. CONCLUSION

Reverse Engineering is the important building block in understanding and maintaining the code. Maintainability increases when the dynamic behavior of the object is translated into design from the source code.

In this paper, we have presented an overview on the reverse engineering of behavioral diagrams. Our approach deals with the reverse engineering from execution traces for object-oriented software. The approach uses a different methodology to deal with the problem of execution traces analyzing. We use for that the CPN. In addition, our approach filter traces. This is very important in the case of GUI systems. It manages also to detect the “par” operator which is very important in the context of multi-threading system.

The future work is to evaluate this approach on more complex system. In addition, we plan to add a new step to our approach. This step include a static analyze of the source code. This is very important to have a more accurate sequence diagram [15]. We will also try to handle the problem to extract a state diagram and other model of the UML diagram [16.17]. Also, the future work will focus on building a CASE Tool to automate the proposed approach and to verify all processes using Petri Net formal method.

REFERENCES

[1] B. Cornelissen, A. Zaidman, et A. Deursen, "A Controlled Experiment for Program Comprehension through Trace Visualization", IEEE Trans. on Software Engineering, 2010

[2] OMG. Unified Modeling Language (OMG UML), Superstructure. V2.1.2. Nov.2007.

[3] Stan Jarzabek and Guosheng Wang, "Model Based Design of Reverse Engineering Tools," Software Maintennance: Research and Practice, J. Softw. Maint: Res. Pract.10, 353-380 (1998).K. Elissa, "Title of paper if known," unpublished.

[4] Elliot j. Chikofsky and James H. Cross II. Reverse engineering and design recovery : a taxonomy. IEEE Software, 1990

[5] A. Rountev, O. Volgin, and Miriam Red-doch. Control flow analysis for reverse engineer-ing of sequence diagrams. Technical Report OSU-CISRC-3/04-TR12, Ohio State University, March 2004.

[6] M. K. Sarkar , T. Chaterjee ; Reverse Engineering : An Analysis of Dynamic Behavior of Object Oriented Programs by Extracting UML Interaction Diagram, International Journal of Computer Technology & Applications, Vol 4 (3) , 2013 ,378-383.

[7] R. Delamare, B. Baudry, Y.L/ Traon "Reverse-engineering of UML 2.0 Sequence Diagrams from Execution Traces" Workshop on Object-Oriented Reengineering at ECOOP 06, Nantes, France, July 2006.

[8] T. Ziadi, M. Aurélio A. da Silva, L. Hillah, M. Ziane, "A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams", 16th IEEE International Conference on Engineering of Complex Computer Systems, 2011, pp. 107-116, (IEEE Computer Society)..

[9] L. C. Briand, Y. Labiche, J. Leduc, "Towards the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software", IEEE Transactions on Software Engineering, vol. 32 (9) , 2006 , pp. 642-663.

[10] R. Zhao, and L. Lin, "An UML Statechart Diagram-Based MM-Path Generation Approach for Object-Oriented Integration Testing",Enformatika; 2006, Vol. 16, p259.

[11] D.H.A. van Zeeland, "Reverse-engineering state machine diagrams from legacy C-code", proceedings of 12th Conf. on Entity-Relationship Approach - Arlington-Dallas, Dec. 1993.

[12] S. Christian, "Extracting n-ary relationships through database reverse engineering", B. Thalheim (Ed.), Proc. 15th Internat. Conf. on Conceptual Modeling, Cottbus, Germany (1996), pp. 392–405.

[13] J.-L. Hainaut, C. Tonneau, M. Joris, M. Chandelon, "Transformation-based database reverse engineering", Lecture Notes in Computer Science, vol. 823, Springer, Berlin, 1994, p. 364.

[14] Michael L. Nelson "A Survey of reverse Engineering and Program comprehension" April 19, 1996.

[15] A. Jakimi, L. Elbermi and M. El Koutbi, "Software Development for UML Scenarios: Design, fusion and code generation". International Review on Computers and Software, Vol. 6. n. 5, pp. 683-687, 2011.

[16] C Baidada, E. Bouziane and A. Jakimi, A New Methodology for Reverse Engineering of UML Behavioral, first international conference of high innovation in computer science, June 01-03, kenitra, morocco, 2016.

[17] M. H. Abidi, A. Jakimi, E. H. El Kinani. A New Approach the Reverse Engineering UML State Machine from Java Code. International Conference on Intelligent Systems and Computer Vision (ISCV'2015), March 25-26 2015, Fes, Morocco.