Towards Interactive Model Mining From Embedded Software

Wasim Said, Jochen Quante

Robert Bosch GmbH, Corporate Research
Renningen, Germany

{Wasim.Said, Jochen.Quante}@de.bosch.com

Abstract

The idea of model mining is the extraction of higher-level models from code. For example, one could extract state machines that describe the behavior of a program. Such models can be very helpful for software maintenance tasks such as program understanding. The major drawback of fully-automatic model extraction is that, when applied on real world systems (e. g., software-intensive systems), the resulting models are complex, difficult, and contain information on a quite low level. Developers can hardly find and understand the information they need. In this paper, we present our ideas towards a framework that solves this problem. The main idea is to let developers contribute to the process of model extraction. The ultimate goal of the framework is to provide high quality representative types of models on an adequate level of abstraction and with low manual effort.

1 Introduction

The extraction of higher-level models can help developers in their task of understanding software systems. The latter is a time-consuming activity that makes up for 40%-50% of total software life cycle effort according to studies [2] and practical experience at Bosch. Model mining can also be a great support for migrating towards model-based software development, which is a trend in the automotive domain. The manual extraction of necessary information from complex software-intensive systems is a tedious and laborious task. Therefore, automation of this process is highly desired. However, fully-automatic model extraction from real world systems results in information on the wrong level: It is usually much too detailed and complex. In the following, we propose an interactive framework that includes expert knowledge and feedback, so that it is capable of extracting high quality models with low manual effort. Our focus is on state machine extraction for now.

2 State machines

State machines model the behavior of software systems through states and transitions between them. They are extensively used in forward engineering in different phases of the development process. In this work, they will be used in reverse engineering to get a good understanding and representation of software system control logic. Most research in state machines extraction has been done with respect to API protocols, i. e., the allowed sequences of API calls [1]. Despite the importance of extracting state machines that describe the *behavior* of an application, little work has been done in this field. Furthermore, a majority of approaches are based on dynamic analysis, which is hardly applicable for real time systems.

Sen and Mall [6] have published an approach to statically extract state machines from Java programs. States are defined as a partition over field values and transitions as the changes on them. This approach comes closest to the objectives of our work. However, it has only been applied on small Java systems, but not on large real world C systems. We have adapted and implemented this approach for C and tried it on our systems. Our experiments show that it fails to extract useful models from these systems: They are too complex and yet incomplete. Therefore, our goal is to make this approach applicable and useful in practice.

3 Interaction with experts

The reason why automatically mined models are too detailed and low-level is that code alone does not contain all the necessary information that would be required. For example, there is no information about which details are important and which are not. Also, the tool is not capable of introducing abstractions that a human would immediately come up with. In this section, we introduce reflexion analysis [4, 3] as an example to illustrate the idea of interaction with experts. Reflexion analysis allows the experts to take part in the analysis process to reconstruct the architecture of the analyzed system. The expert starts by providing a hypothetical model that represents important components and expected dependencies between them. This model is called the conceptual model. The expert also provides a mapping between his conceptual model and implementation artefacts in system. The reflexion analysis then examines automatically which dependencies are present in the code and lifts them to the conceptual model. It compares the specified dependencies from the expert with the existing dependencies in the code, then it gives feedback to the expert to refine his conceptual model or his mapping, and the process starts over again.

This approach has been proven as being very useful for extracting architectural models from software systems [3]. The basic idea could therefore also be applied to extract other models or information from software systems.

4 Analysis

Figure 1 shows how the model could be interactively extracted from legacy code. We have used different colors to show that some steps are manual, semi-automatic or automatic. For example, the engagement of the user is a manual procedure, whereas the tool extracts the states automatically. The final state machine models will then be semi-automatically extracted.

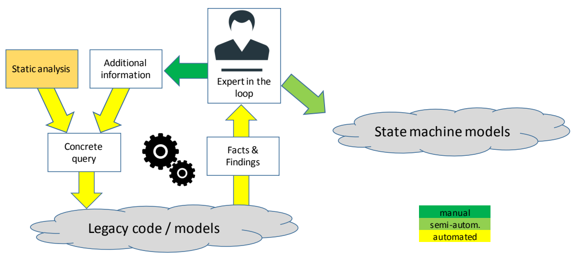

Figure 1: The extraction of state machine models through the interaction between tool and expert.

In the following, we will discuss different possible scenarios for interacting with experts, which can make this approach applicable on real-world software and not only on simplistic examples, as is the case for available approaches.

1. Bottom up: In this scenario, the expert has no idea about specific states or modes, and he asks for general information. The tool should extract state candidates from code and display them to the user, ideally sorted according to some relevant criteria. The user then selects the states of interest. This early interaction phase leads to a reduced state space and thereby smaller models. The process proceeds with iterations and interactions with user. Consequently, the user can get multiple small and easy to understand state machines instead of only large and complex one.

2. Top down: If the user has a hypothetical state machine model in mind that he expects to be present in the system, we can apply the idea of reflexion analysis. The state space in this case is only what the user model contains. Thus, the tool can directly start collecting information about the specific states and transitions from source code. The user receives feedback about which parts in the hypothetic model match the real system and which parts do not. He can then refine his model and iteratively obtain more information about his hypothesis.

In both scenarios, the expert could be interested in some more specific situations that the system could be in. For example, he could define a specific invariant such as speed=0. In this case, the tool should provide only states that the system has when speed=0. All other information related to speed>0 or speed<0 should then be hidden. This step helps the expert to get even more understandable results (models), because the state space becomes smaller and simpler – and the expert can focus on the relevant information.

5 First Results

Based on the approach of Sen and before mentioned bottom up scenario, an initial approach was implemented to extract state machines interactively with experts. The implementation is mainly based on the SWAN Software Analysis Framework, which has been developed at Bosch. It contains a generic model interpreter [5] and tools for data and control flow analysis, model reduction and visualization. The first results are very promising. The approach performed well by responding to user choices and by extracting the reduced state machines. It also highlighted some challenges in the process of extracting understandable models from real world systems, such as the simplification of transition conditions.

6 Outlook

We have presented the first steps towards extracting understandable state machines interactively from embedded software. We will next implement different ideas for interactive model mining. After that, we will empirically check whether the approach really supports program understanding.

References

[1] G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. *SIGPLAN Not.*, 37(1):4–16, Jan. 2002.

[2] R. K. Fjeldstad and W. T. Hamlen. Application program maintenance study: Report to our respondents. In *Proc. GUIDE 48*, Apr. 1984.

[3] R. Koschke and D. Simon. Hierarchical reflexion models. In *Proc. of 10th Working Conference on Reverse Engineering*, WCRE '03, pages 36–45, USA, 2003.

[4] G. C. Murphy, D. Notkin, and K. Sullivan. Software reflexion models: Bridging the gap between source and high-level models. *SIGSOFT Softw. Eng. Notes*, 20(4):18–28, Oct. 1995.

[5] J. Quante. A program interpreter for arbitrary abstractions. *16th Int'l Working Conference on Source Code Analysis and Manipulation*, pages 91–96, 2016.

[6] T. Sen and R. Mall. Extracting finite state representation of java programs. *Software & Systems Modeling*, 15(2):497–511, 2016.