# Interactive Model Mining from Embedded Legacy Software

Wasim Said
Robert Bosch GmbH, Corporate Research, Renningen, Germany
University of Bremen, Germany
wasim.said@de.bosch.com

## ABSTRACT

Model mining from software systems can be very helpful for program comprehension. The few existing approaches for extracting high level models from code – when applied to real-world systems written in C – deliver too detailed and complex models that cannot be understood by humans. In my Ph.D. project, I propose an approach that complements fully-automatic model mining approaches with user interaction to get understandable models. The evaluation of this approach includes a controlled experiment with a large number of experts, in order to assess the effectiveness of the interactively mined models for understanding complex legacy software.

**ACM Reference Format:**
Wasim Said. 2018. Interactive Model Mining from Embedded Legacy Software. In *ICSE '18 Companion: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183440.3183448

## 1 INTRODUCTION

Embedded legacy software contains accumulated expert knowledge from different people and domains, which makes it valuable and complex at the same time. Understanding these systems is required to maintain and evolve them. However, program comprehension is a time-consuming activity that makes up for about 50% of total software life cycle effort [2, 7].

Model mining is the extraction of behavioural and/or structural models from software systems. Having such higher-level models can help developers in a) program understanding and software activities based on it, such as maintenance, debugging, validation and verification and b) migration towards model-based software development, which is a trend, e.g., in the automotive industry.

The manual extraction of helpful models from software-intensive systems provides very understandable models. On the other hand, it is error-prone, tedious and laborious. Therefore, automation of this process is highly desired. However, fully-automatic model mining from complex real world systems results in too detailed and low-level models, which cannot be understood by human experts and do not satisfy experts demands of understanding specific aspects of the code. This is because code alone does not contain all the necessary information. For example, there is no information about which details are important and which are not. Also, a tool is not capable of introducing abstractions that a human would immediately come up with. Therefore, human interaction is essential.

In my thesis, I develop an **interactive** approach for model mining from embedded legacy software. This approach utilizes expert knowledge and feedback to extract useful models with low manual effort. I claim that the combination between tool and expert will be very effective, since it combines the strengths from both sides: A tool is very strong in evaluating complex dependencies, finding patterns in large data sets, considering all details at once and providing consistent transformations. The expert, on the other side, can very well assess the relevance of individual artefacts, define the right level of generalization/specialization, give meaningful abstractions and know the relation to the environment. Therefore, the first research questions are:

(1) **RQ1:** Does the interactive approach extract more understandable models than fully-automatic approaches?
(2) **RQ2:** Regarding the comprehensibility aspect: Can the interactively mined models be of the same high quality as the manually extracted ones?

In the proposed approach, the tool is responsible for software analysis. The existing software analysis techniques suffer from different limitations. For example: dynamic analysis extracts information only from the executed traces, which makes it too concrete, incomplete and difficult to generalize. It is also very difficult to find representative test cases. In addition, embedded systems and especially automotive systems – on which I will focus in my Ph.D. project – are real time systems with strict time constraints. They cannot easily be instrumented for dynamic analysis, because this would alter runtime effects (in particular timing) in a way that completely changes the system's behavior. Approaches based on natural language analysis require not only well defined requirement documents, but also these documents have to be kept up to date, which is almost always not the case in legacy systems. Other approaches, such as evolution analysis or repository mining focus on the changes of code. In my case, I need to understand the basics of the code and not only the clones or the changes.

Therefore, static analysis is the most appropriate technique for model mining from embedded software. However, the main drawback of static analysis is that it is too conservative and delivers all details including infeasible cases,


*ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden*

ACM ISBN 978-1-4503-5663-3/18/05...$15.00
https://doi.org/10.1145/3183440.3183448

which makes the mined models very complex and therefore useless. I claim again, that user interaction can also overcome static analysis drawbacks: When the user can determine the relevant information and abstract away everything irrelevant, the desired models are extracted faster and will be more understandable. This leads to the next research questions:

(3) **RQ3:** Can the user interaction control the static analysis process in a way that it results in understandable models?
(4) **RQ4:** Do the additional interactions pay off with respect to time and effort and are they accepted by users?

Embedded systems are usually implemented in C (or ASCET in automotive systems), which makes a lot of model mining approaches not applicable to them. For example, most state machine mining approaches deal with object-oriented systems. The few existing approaches for procedural (not object oriented) code expect specific implementation patterns in the code, such as switch-case constructs on enums. These approaches quickly reach their limits when the code is not implemented based on these patterns. Therefore, the next research questions are:

(5) **RQ5:** What techniques can be adapted and applied from the existing model mining approaches for embedded software written in procedural languages?
(6) **RQ6:** Which limitations of these approaches require new techniques, what are these techniques and how can they be implemented and evaluated?

Different kinds of models can be extracted from embedded legacy software. Each kind is useful in specific fields, such as state machines and variance models of software product lines. This point raises the next research question:

(7) **RQ7:** What are the common parts of the proposed interactive model mining approach that can be reused to extract different kinds of models and which parts are model-dependent?

## 2 METHODOLOGY

In this section, I discuss how I intend to answer the research questions:

(1) **RQ5, 6:** At the beginning, I researched the state of the art approaches for state machine mining. Then, I adapted a fully-automatic static analysis approach – that extracts state machine models from code – for application to procedural languages. In the next step, I applied the selected approach on several real-world functions from the automotive domain and assessed the mined models, in order to determine the weak points in them. Based on this case study, I defined different user interaction scenarios, which a) can be combined with static analysis techniques, b) are able to overcome the determined drawbacks and c) can make the mined models more understandable. I implemented these interactions with static analysis techniques in a prototype, which can interactively extract state machine models from C code.
(2) **RQ1, 2, 3, 4:** To evaluate the proposed interactive approach, I plan to apply the prototype to many industrial software projects with functions that are classified by developers as difficult to understand. These functions are written by a large number of developers and taken from different business units. I will evaluate each interaction scenario separately and in combination with other scenarios to find out which interaction is most helpful given different situations. I also want to assess the developers' effort for applying the various selection and filtering features. In addition, because only humans who work with the code can determine how helpful and understandable are the extracted models, I plan to conduct a controlled experiment with domain experts and function responsibles to evaluate the understandability of the mined models. There is already a strong interest from Bosch experts in this experiment.
(3) **RQ7:** In order to determine whether the interactive approach can be reused to extract other kinds of models from real world systems, I plan to exploit the gained experience from state machine mining to extract other kinds of models. An example thereof will be variance models. In this phase, I will determine the parts of the interactive approach that can be directly reused and the parts that must be processed according to the extracted kind of models.

## 3 STAGE OF THE RESEARCH AND TIMELINE

I have spent one and a half year so far working on interactive state machine mining from embedded software. I implemented an initial approach to extract state machines from C code and ASCET models interactively with experts. My contributions till now with the time required for research, design and implementation are listed as follows:

(1) **Adaptation of state machine mining approaches:** My approach is implemented based on the approaches of Sen and Mall [13] and Kung et al. [9] with slightly different settings, adaptations and optimizations. For example, I use concolic testing [3] instead of symbolic execution [6] used by Kung and Sen. The differences between symbolic execution and concolic testing are discussed in [4]. (RQ5 - 3 months)
(2) **Identification of state variables from C code:** The approaches of Kung and Sen both work on object-oriented software, whereas the systems that I deal with are written in C or ASCET. State variables by Kung/Sen approaches are simply the member variables of the given class. In C code, there are no natural member variables. Therefore I defined criteria to determine the relevant state candidate variables in procedural code and I conducted a case study with domain experts to validate the proposed criteria. (RQ6 - 3 months)

(3) **Transitions and transition conditions:** I refined the transition extraction of state machine models, which led to a reduction of the number of transitions by more than 50% on average by removing the infeasible ones. I evaluated several approaches, such as BDDs, to reduce the complexity of transition conditions (conditions that must be fulfilled to trigger a transition between two states), and I did a survey to determine the characteristics of understandable transition conditions. (RQ6 - 3 months)

(4) **Interactive exploration:** I defined a set of user interactions and applied them to mine state machine models from embedded software. For example, selecting a subset of state variables, selecting certain states or merging states, setting variables to certain values and adding constraints to the mining process. These interactions can be applied separately or in any combination, which give the user a rich set of possibilities for exploring the behaviour of the function. (RQ1, 2, 3 - 3 months)

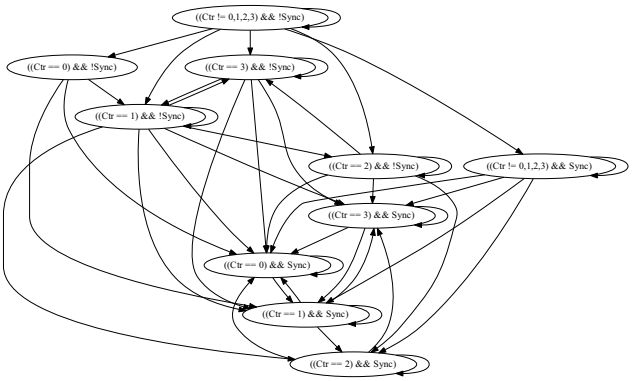
(5) **First evaluation and feedback:** I evaluated the effectiveness of these interactive measures on making the resulting state machines understandable, and I did a first assessment with experts about the usefulness of the extracted state machines. The first results are very promising: My case study showed that the user interactions can strongly reduce the complexity of the extracted state machines. The first interviews with experts suggest that the reduced models are still quite helpful for program comprehension, migration towards models and other software activities, such as validation, verification and debugging. (RQ1, 2, 3, 4 - 3 months)

The implemented state machine mining approach (1), the interactive exploration (4) and the first evaluation (5) are published in [12].

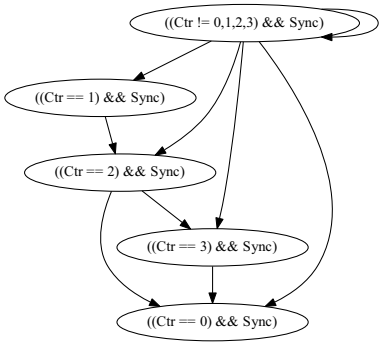
Figure 1 shows an example from my experiments about the proposed user interactions and how they can make the state machines more understandable. The subject function is a real world function of an engine control software. It contains 6 state variables that generate 720 states and 1,652 transitions between them, which make this model too complex. Figure 1(a) shows the extracted state machine when the user selected only two state variables. In Figure 1(b), we see the model of the same function – with the same two state variables – when the user manually added 12 constraints on non-state variables in code, with help from the information that was presented to him by the tool. This model contains a lower number of states and transitions and therefore it is more understandable than the first one.

My future work includes:

(1) **Interactive state machine mining:** I will continue my work on interactive state machine mining to add more user interactions and solve the problems that appeared in the approach. For example, despite using different approaches to reduce transition conditions of state machines, some of these conditions are still complex for human comprehension. Therefore, I will apply another interactive extension to make these conditions better understandable. Another problem is the computation time: Because my approach uses an SMT solver in concolic testing, the computation time is quite high. The different interactive scenarios have strongly reduced the required time to extract the models. However, I plan to do more optimizations for the computation time. (RQ5, 6 - 6 months)




(a) The model with only two state variables.




(b) The same model with user constraints.

**Figure 1: The effect of user interactions on state machine mining.**

(2) **Comprehensive evaluation of the approach with controlled experiment**: see Section 2 point 2. (RQ1, 2, 3, 4 - 6 months)

(3) **Form the basis of a general interactive model mining approach**: see Section 2 point 3. (RQ7 - 6 months)

## 4 STATE OF THE ART

Little work has been done in the extraction of state machines that describe the behavior of an application. I will focus on these approaches. Kung et al. [9] and Sen and Mall [13] extract state machines automatically from C++ source code and Java programms, respectively. My experiments with these approaches showed that they fail to extract understandable

models from typical embedded software: Even small functions can result in hundreds of states and transitions.

Corbett et al. [1] automatically extract state machines from the source code of Java programs for model checking. The generated models are in the input language of one of the used verification tools. The resulting models with thousands of states are not a problem for this use case – but for human comprehension, they are. Some approaches can only be used for specific cases such as the work of Van den Brand et al. [14]. They expect specific patterns in code such as nested-choice patterns (e. g., nested `if`s). The approach is restricted to these patterns and thus not applicable to the more general kind of systems that I am interested in.

Regarding the interaction with experts, many previous studies such as [11] and [5] suggest that the process of extracting understandable models must be human-guided, but no one ever followed up on this idea for state machines. Reflexion analysis [10] is an example of the interaction with experts. It allows the experts to take part in the analysis process to reconstruct the architecture of the analyzed system. The expert provides 1) a conceptual/hypothetical model that represents important components and expected dependencies between them, and 2) a mapping between this 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 ones in the code, then it gives feedback to the expert to refine his conceptual model or mapping, and the process starts over again. This approach has been proven as being very useful for extracting architectural models from software systems [8].

## 5 EXPECTED OUTCOME

At the end of this project, I expect to have a framework for comprehensive interactive state machine mining. The framework will interact with the user to get the desired models quickly. In addition, the framework will also support the extraction of other kinds of models. The expected use cases for the interactive model mining framework, based on my experiments and first feedback from experts, are:

**Program understanding:** Interactive model mining can meet the needs of different users with different backgrounds about the system. In my experiments with state machines, the developers were able to understand the models and determine the main ideas quickly. The interaction scenarios can also be used for specific purposes like investigating what-if scenarios, e. g., by setting some variables to certain values, which can strongly help in understanding the function under consideration.

The following example from my experiments shows how the framework can be helpful for **validation and verification**: An expert, who was responsible for a subject function, thought that some transitions in the extracted state machines would not exist in the code. However, when he reviewed the code, it turned out that the transitions do exist.

The proposed framework can be a great support for **Model-Driven Engineering**: Companies that want to switch to model-based development usually do not start from scratch, but already have a large code base. Interactive model mining can support the transformation from code to models and even make it economically worthwhile.

I also have the plan to test how the interactive mining of models can support users in other software engineering activities, such as model checking, simulation and user-interface design.

In summary, adding user interaction to model mining seems to be a promising approach towards helpful support for program comprehension and migration towards model-based development. The framework with the comprehensive evalution is expected to be finished in summer 2019.

## REFERENCES


[1] James C. Corbett, Matthew B. Dwyer, John Hatcliff, Shawn Laubach, Corina S. Pasareanu, Robby, and Hongjun Zheng. 2000. Bandera: extracting finite-state models from Java source code. In *Proc. of 22nd ICSE*. 439–448.

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

[3] Patrice Godefroid, Nils Klarlund, and Koushik Sen. 2005. DART: directed automated random testing. In *Proc. of PLDI*. 213–223.

[4] Andreas Hoffmann, Jochen Quante, and Matthias Woehrle. 2016. Experience Report: White Box Test Case Generation for Automotive Embedded Software. In *Proc. of 9th Int'l Conf. on Software Testing, Verification and Validation Workshops, TAIC-PART Workshop*. 269–274.

[5] Rahul Jiresal, Hemanth Makkapati, and Ravindra Naik. 2011. Statechart Extraction from Code – An Approach using Static Program Analysis and Heuristics Based Abstractions. In *Proc. of 2nd India Workshop on Reverse Engineering*.

[6] James C. King. 1976. Symbolic execution and program testing. *J. ACM* 19, 7 (1976), 385–394.

[7] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung. 2006. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *IEEE Transactions on Software Engineering* 32, 12 (Dec 2006), 971–987.

[8] Rainer Koschke and Daniel Simon. 2003. Hierarchical Reflexion Models. In *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE '03)*. 36–45.

[9] David Chenho Kung, N. Suchak, Jerry Zeyu Gao, Pei Hsia, Yasufumi Toyoshima, and Chris Chen. 1994. On object state testing. In *Proc. of 18th Int'l Computer Software and Applications Conference (COMPSAC)*. 222–227.

[10] Gail C. Murphy, David Notkin, and Kevin Sullivan. 1995. Software Reflexion Models: Bridging the Gap Between Source and High-level Models. *SIGSOFT Softw. Eng. Notes* 20, 4 (Oct. 1995), 18–28.

[11] N. Prywes and P. Rehmet. 1996. Recovery of software design, state-machines and specifications from source code. In *Proc. of 2nd Int'l Conf. on Engineering of Complex Computer Systems*. 279–288.

[12] Wasim Said, Jochen Quante, and Rainer Koschke. 2018. Towards Interactive Mining of Understandable State Machine Models from Embedded Software. In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD*. 117–128.

[13] Tamal Sen and Rajib Mall. 2016. Extracting finite state representation of Java programs. *Software & Systems Modeling* 15, 2 (2016), 497–511.

[14] Mark van den Brand, Alexander Serebrenik, and Dennie van Zeeland. 2008. Extraction of State Machines of Legacy C code with Cpp2XMI. In *Proc. of 7th Belgian-Netherlands Software Evolution Workshop*. 28–30.