Extraction of state machines of legacy C code with Cpp2XMI

Extraction of State Machines of Legacy C code with Cpp2XMI

Mark van den Brand, Alexander Serebrenik, and Dennie van Zeeland

Technical University Eindhoven, Department of Mathematics and Computer Science,
Den Dolech 2, NL-5612 AZ Eindhoven, The Netherlands
m.g.j.v.d.brand@tue.nl, a.serebrenik@tue.nl,
d.h.a.v.zeeland@student.tue.nl

Introduction Analysis of legacy code is often focussed on extracting either metrics or relations, e.g. call relations or structure relations. For object-oriented programs, e.g. Java or C++ code, such relations are commonly represented as UML diagrams: e.g., such tools as Columbus [1] and Cpp2XMI [2] are capable of extracting from the C++ code UML class, and UML class, sequence and activity diagrams, respectively.

New challenges in UML diagram extraction arise when a) additional UML diagrams and b) non-object-oriented programs are considered. In this paper we present an ongoing work on extracting state machines from the legacy C code, motivated by the popularity of state machine models in embedded software [3]. To validate the approach we consider an approximately ten-years old embedded system provided by the industrial partner. The system lacks up-to-date documentation and is reportedly hard to maintain.

Approach We start by observing that in their simplest form UML state machines contain nothing but states and transitions connecting states, such that transitions are associated with events and guards. At each moment of time the system can be in one and only one of the states. When an event occurs the system should check whether the guard is satisfied, and, should this be the case, move to the subsequent state. Observe, that implementing a state machine behaviour involves, therefore, a three-phase decision making:

- What is the current state of the system?
- What is the most recent event to be handled?
- Is the guard satisfied?

Based on this simple observation, our approach consists in looking for *nested-choice patterns*, such as "`if` within `if`" or "`switch` within `switch`". As guards can be omitted we require the nesting to be at least two. As we do not aim to discover all possible state-machines present in the code, validation of the approach will consist in applying in the case study and comparing the state-machines detected with the results expected by the domain experts.

Implementation We have chosen to implement the approach based on the Cpp2XMI tool set [2]. Since Cpp2XMI was designed for reverse engineering C++, we first had to adapt the tool for C. Second, we added a number of new filters to detect the nested-choice patterns in the abstract syntax trees. Finally, we had to extend the visualisation component to provide for state machine visualisation.

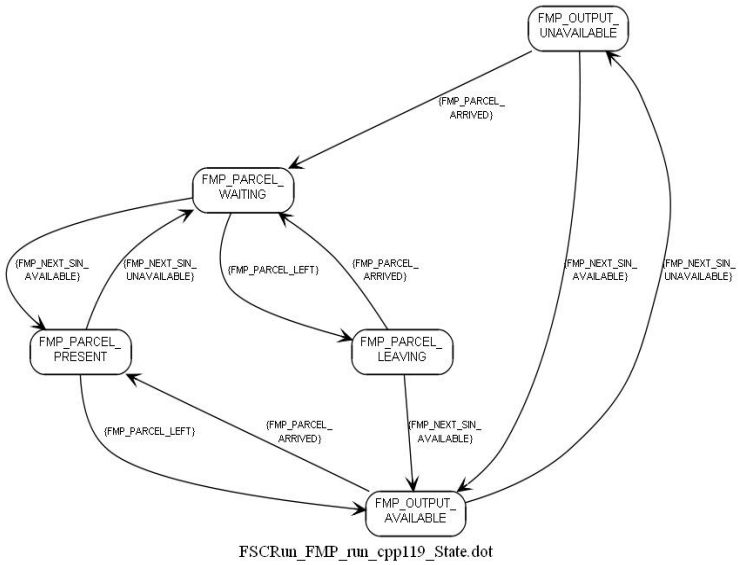

Fig. 1. A state machine discovered.

Case study As the case study we consider an approximately ten-year old system, developed for controlling material handling components, such as conveyer belts, sensors, sorters, etc. Up-to-date documentation is missing and the code is reportedly hard to maintain. While a re-implementation of the system is considered by the company, understanding the current functionality is still a necessity.

It turned out that the original software engineers have quite consistently used `switch` statements within `switch` statements to model the state machines. Therefore, already the first version of the implementation based solely on the "`switch` within `switch`" pattern produced a number of relevant state machines.

At the moment more than forty state machines have been extracted from the code. The size of the extracted state machines varied from 4 states up to 25 states. One of the extracted state machines is shown on Figure 1, the transitions are decorated with conditional events. All the machines extracted were presented to the (software) engineers of the company and their correctness as well as importance were confirmed by them.

Conclusions and future work. In this abstract we presented an ongoing effort on extracting UML state machines from legacy non-object-oriented code. We have observed that UML state machines are useful for the developers and maintainers, and that they can be derived automatically even from a non-object-oriented code. The approach proved to be very successful in the case study and, is in general, promising. As the future work we consider:

- including the "`switch` within `if`" and "`if` within `switch`" patterns;
- analysing the extracted state machines for overlap;
- combining the extracted state machines to nested state machines.

References

1. Rudolf Ferenc, Árpád Beszédes, Mikko Tarkiainen, and Tibor Gyimóthy. Columbus - reverse engineering tool and schema for c++. In *ICSM*, pages 172–181. IEEE Computer Society, 2002.
2. E. Korshunova, M. Petkovic, M. G. J. van den Brand, and M. R. Mousavi. Cpp2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code. In *WCRE*, pages 297–298. IEEE Computer Society, 2006.
3. Jürgen Mottok, Frank Schiller, Thomas Völkl, and Thomas Zeitler. A concept for a safe realization of a state machine in embedded automotive applications. In Francesca Saglietti and Norbert Oster, editors, *SAFECOMP*, volume 4680 of *Lecture Notes in Computer Science*, pages 283–288. Springer, 2007.