4L−7

An Approach to Model-Based Construction of Soft Real-Time Embedded Java Code

Ilankaikone Senthooran[†*]   Takuo Watanabe[†]
[†]Department of Computer Science, Tokyo Institute of Technology

Abstract

Developing software for embedded real-time systems has become a challenge while maintaining its functional requirements, such as timeliness, correctness, etc. Model-based formal verification method is widely applied in the code development of real-time systems. However, verification of such model only ensures the correctness of the model, not the software of that system. We propose a method for generating Java code enabling soft real-time feature, from a model verified by UPPAAL model checker, which is based on Timed Automata. Thereby increasing confidence in the code, based on the notion of correct-by-construction. Finally, we evaluate our method by applying it to non-trivial real-time systems, including a maze-solver robot.

1 Introduction

Embedded software creates both huge value and unprecedented risks due to the complex system context of embedded software applications. Real-time software differs significantly from the conventional software[1]

This paper presents a systematic approach to generate working software code for real-time embedded systems, see Figure 1. We adopt timed automata [2] as design models (modeling language) for timed systems, and study how to transform such models to Java code. There are several tools developed to verify models, of which we use UPPAAL [3] Model-Checker extends timed automata with several additional features such as urgent locations, integer and arrays.

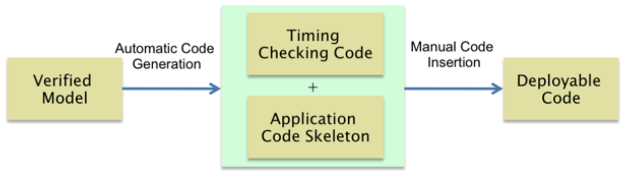

Figure 1: Our Approach

*senthooran@psg.cs.titech.ac.jp

Once the model is created, it undergoes verification[4][5] process in UPPAAL. Then the program code is generated from the model. However, generated code cannot be applied straight away, because it represents the abstract model of the system. The systems variables are added, to make the program code deployable in a given environment.

Our Contribution We developed a method to convert real-time systems modeled in Timed Automata to Java[6] program code mingled with timing checking code, enabling *soft real-time* features. This code can be deployed in many general embedded devices as the code is not RTS Java.

1.1 Related Work

Iftikhar et al [9] presented an approach for Java code generation from UML statechart. However, UML statecharts do not formally support verification methods.

Niusha et al [10] presented as approach for real-time specification Java from Timed Automata. Their approach uses native real-time features.

2 Systematic Code Generation

To achieve a systematic code generation, we propose rules to convert the different features of UPPAAL Timed Automata into the Java Code. Then, collating these code fragments according to the model. Features of UPPAAL Timed Automata are: *Timed Automaton, Binary Synchronization, Broadcast Channel, Location (or State), Clock, Guard, Invariants, Global Variable, Urgent Synchronization, Urgent / Committed Location.*

Every Timed Automaton is converted to a Java thread. The behavior of the automaton is encoded inside the thread `run` method within an infinite while loop. Likewise, rest of the features are converted to Java code. Mapping the features of the UPPAAL model to the code with appropriate rules will enable us to create program code, which matches modeled behavior. The notion is called `correct-by-construction`.

We have not dealt translating features `Urgent Binary Synchronization, Urgent / Committed Location` to Java code, as the time is not allowed to pass. Therefore, we have considered these features as normal binary synchronization or normal location and applied the normal conversion rules to generate Java code.

2.1 Non-determinism

A state in a Timed Automaton may contain more than one enabled outgoing edge. So, the converted program will always perform the first state change. This means that the translated code is deterministic. Following the perception that an implementation should be more deterministic, we propose an ordering method based on the guard `clocks`. To handle ordering of other scenarios, we directly use the ordering present in the model.

3 Case Study

We applied our rules on a Line Maze-Solver to validate our approach, where a robot will run through a lined-maze to find the shortest path to the target. We constructed the robot using Java SunSPOT [7], and Pololu 3pi Robot [8]. SunSPOT serves as the controller for the 3pi robot, its functions are learning the maze, finding shortest route and directing the 3pi robot to the destination. SunSPOT and 3pi communicates by means of IO pins.

3.1 Results

The final model contains 2 Timed Automatons with a total of 23 states and 38 transitions. The generated code had 2 Java Threads, around 500 lines of code and 8 channels. The final deployable code amounts to 800 lines.

We encountered a practical problem during test run, where the robot did not take turns exactly at the intersection, due to communication delay in the hardware, and we a made small change in the model to tackle this issue. Based on our results, the robot behaved as per to the model. We repeated the process from various points on the maze and on every run eventually robot was guided to the target.

4 Conclusion

In this paper, we tried to arrive at a proper code conversion rules for *soft real-time* systems that does not violate the properties of the model of the system. We applied our approach on a real-time example: maze-solver. Based on our experience, the results were promising, but we had to make some changes in the model due to some hardware limitations. Also, we realized that our approach could be extended to a C programmable device. However, our approach cannot tackle specific time actions as it is not practically achievable in a non real-time Java based platform.

As our future work, the manual code insertion will be partially automated by attaching a module as input to the code generator, which contains the replacing code fragments, while reducing potential human error.

References

[1] Alan C. Shaw, *Real-Time Systems and Software.* John Wiley & Sons, New York, 2001.

[2] Rajeev Alurand DavidL.Dill, *A theory of timed automata.* Theoretical Computer Science, 126(2):183-235, 1994.

[3] Kim G. Larsen, Paul Pettersson, and Wang Yi, *UPPAAL in a nutshell.* International Journal on Software Tools for Technology Transfer, 1(1-2):134-152, 1997.

[4] Farn Wang, *Formal Verification of Timed Systems: A Survey and Perspective.* Proceedings of the IEEE, 2004.

[5] Jr. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled, *Model Checking.* MIT Press, 1999.

[6] Andy Wellings, *Concurrent and Real-Time Programming in Java.* John Wiley and Sons Ltd, 1994.

[7] *Sun SPOT World.* Available at `http://sunspotworld.com/docs/index.html`, [accessed 12 July 2010].

[8] *Pololu 3pi Robot User?s Guide.* Available at `http://www.pololu.com/docs/0J21`, [accessed 12 July 2010].

[9] lIftikhar Azim Niaz and Jiro Tanaka, *Mapping Uml Statecharts To Java Code.* Proc. IASTED International Conference on Software Engineering, 111-116, Innsbruck, Austria, 2004.

[10] Niusha Hakimipour et al, *Exploring Model-Based Development for the Verification of Real-Time Java Code.* IJCAR08-VERIFY08, Sydney, Australia, 2008