A Lightweight Development Environment for Embedded System with Go Language

Sari Orioka[1] Hiroyuki Okamura[1,a)] Tadashi Dohi[1,b)]

Abstract: This paper presents a Go-language-based lightweight development environment for embedded system. The features of our tools are (i) open source license, (ii) generating source codes from state machine diagrams, (iii) concurrent simulations for system verification. In particular, by combining existing tools, we also realize the functionalities for drawing state machine diagrams, compiling binaries for embedded system and writing binaries into hardware.

1. Introduction

Embedded systems are computer systems that are equipped in electrical appliances and automobiles to control themselves. The development of embedded system is generally divided into five phases: system design, software design, program implementation, software testing, and system verification. Since the embedded system interacts our daily lives through the physical devices, it requires high reliability and safety. Also, the embedded system runs on the specific physical devices, and thus it is not easy to remove bugs in the system testing phase. Therefore, to ensure high reliability and safety, it is important to remove defects and faults in the early phase of system development. In particular, it is known that it is effective to build a model representing the system behavior in the system design phase, and the model also enables us to verify the system meets requirements or not.

A state machine (SM) is one of the modeling approaches to represent the system behavior. The SM consists of state and transitions. The state determines the action that should be done, and the transition provides the conditions to change the state. The SM diagram is the representation that defines states and transitions of an SM, and is also included in SysML (systems modeling language). Since the SM diagram helps us to verify the system behavior well, it is commonly used in the system design of embedded system [1].

This paper presents a lightweight integrated development environment with Go language for embedded systems. In particular, we provide the tools to generate source codes from a given SM diagram, and to simulate the system behavior in software.

2. Overview of tools

Our tools support the following processes in the development of embedded system;

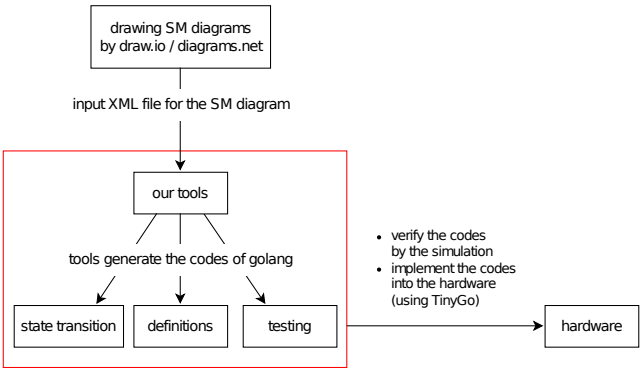

Fig. 1 the flow of the system development with our tools

(i) drawing SM diagrams,
(ii) implementation of software codes,
(iii) verification of software codes; and
(iv) implementation to hardware.

Figure 1 shows the flow of the system development with our tools. In the figure, the red rectangle presents the processes (ii) and (iii) that are related to the tools developed by ourselves. The other processes (i) and (iv) are realized with the existing tools 'diagrams.net' and 'tinyGo'.

(i) drawing SM diagrams

The SM diagram is created by an existing application called diagrams.net/draw.io. The diagrams.net/draw.io is an open source for building diagramming applications, and provides a browser-based software. The diagram in application is an XML-based document. We provide a template for diagrams.net/draw.io to create the SM diagram. The template includes the information to convert it to the source code beforehand. By using the template, we can draw the diagrams that can generate the codes with our tool.

(ii) implementation of software codes

From the XML file for the SM diagram in (i), we can generate the codes of Go language (golang). The golang is a programming language by Google. The tool parses the XML file, and gener-

[1] Graduate School of Advanced Science and Engineering, Hiroshima Uniersity, 1–4–1 Kagamiyama, Higashi Hiroshima, 7398527, Japan
[a)] okamu@hiroshima-u.ac.jp
[b)] dohi@hiroshima-u.ac.jp

Table 1 Comparison with existing tools.

	astah* professional	Enterprise Architect	Our tools
Languages	C++, Java, C#, etc	C, C++, Java, C#, VB.NET	golang
Simulation	Static state transition paths	Dynamic state transitions	Concurrent simulation
Lisence	Commercial	Commercial	Open license

ates (a) state transition, (b) definitions and (c) testing. The state transition (a) provides the codes to change the state by triggering transitions. The actions on state and the conditions for transitions are defined in (b) definitions. The tool provides a skeleton for the action and condition codes. The concrete codes will be written according to the skeleton by hands. The testing (c) is the example code to run the simulation according to the SM diagram. In particular, the definitions of actions follow the SysML manner, i.e., each state has the actions called entry, do, and exit, and these actions are executed by state transitions. The entry action is the program running on the beginning of state once, the exit action is the program running on the ending of state once. The do action is the program running during the state continuously.

(iii) verification of software codes

The verification of codes is done by the simulation. The simulation means the execution of SM based on some scenarios in software. By monitoring simulation paths, we can verify whether the SM correctly runs or not. In our environment, the simulation execution is given by a test code of golang. By following golang-based testing manner, the simulation runs with one command 'go test' simply. Also, the simulation for the embedded system consists of user behavior and system behavior, and these behaviors are parallel in real situation. To simulate such situation, our environment utilizes Goroutine. Goroutines are lightweight threads managed by golang. Compared to the ordinary threads, they have the features; low memory usage and low switching cost (overhead). In addition, Goroutine can easily be used in golang programs. By using this functionality, our environment can easily create threads to simulate system and user behaviors concurrently.

(iv) implementation to hardware

To implement the codes into the hardware, we use TinyGo [3]. TinyGo is an open source project to compile binaries from golang source codes for embedded systems such as Arduino. In addition, TinyGo includes a tool set to write the binary into such hardware. TinyGo is a subset version of the original golang for embedded systems. The original golang depends heavily on POSIX-OS and requires a rich runtime, and thus it is not suitable for embedded system. On the other hand, TinyGo is a compiler of the golang that supports embedded architectures using LLVM (a framework for creating compilers) while substituting POSIX-OS-dependent functions with a simple implementation. It can run programs on more than 85 microcontrollers, including the BBC micro:bit, Arduino Uno and so on.

3. Related tools

The significant features of our tools are (i) creating source codes from SM diagrams, (ii) running concurrent simulations. From these points of view, we compare our tools to existing tools. There are two commercial tools to create source codes from SM diagrams; astah* professional [4] and Enterprise Architect (EA) [5]. Table 1 presents the comparative results for the functionalities in these tools.

astah* professional can generate code templates for a number of programming languages via plug-ins. On the other hand, it does not support the simulation run of the SM diagram. Instead of that, astah* professional can output static state transition paths of SM diagrams by the plug-in.

EA can also generate code templates for several programming languages from SM diagrams, and can output the state transition paths of SM diagrams dynamically. This is similar to the simulation provided by our tools. However, this is not exactly same as our tools, because user interactions cannot be handled in EA. To verify whether the system meets some requirements, the user behavior should be considered. The state transition simulation is not enough for the system verification, but the concurrent simulation with user and system behaviors is needed.

4. Conclusion

This paper presented the development environment for embedded system with golang. In our tools, by combining existing open source software, we provide an integrated development environment from creating SM diagrams to writing compiled binaries. In particular, it has advantages on the automatic generation of source codes from SM diagrams and the concurrent simulation frameworks for system verification. This is expected to enable us to develop highly-reliable embedded systems with low efforts.

In the current version, the tool can handle the code generation from the SM diagrams without composite states and orthogonal states. In future, we plan to enhance our tools so that it can handle SM diagrams with composite states and orthogonal states. In addition, under our testing framework, the system verification is not automatic but we have to check the behaviors by hands. Thus we implement the automatic verification by combining model checking techniques.

References

[1] Takayuki, M.: State Transition Design Methodology for Embedded Engineers, TechShare Corporation(2016).
[2] diagrams.net, https://app.diagrams.net/
[3] TinyGo, https://tinygo.org/
[4] astah* professional, https://astah.change-vision.com/ja/product/astah-professional.html
[5] Enterprise Architect, https://www.sparxsystems.jp/