Formal Verification of Robot Movements
- a Case Study on Home Service Robot SHR100

Moonzoo Kim, Kyo Chul Kang
Computer Science and Engineering Department
Pohang University of Science and Technology
Pohang, South Korea
{moonzoo,kck}@postech.ac.kr

Hyoungki Lee
Interaction Lab.
Samsung Advanced Institute of Technology
Suwon, South Korea
twinclee@samsung.com

Abstract—Home service robots have received much attention from both academia and industry because home service robots have wide range of potential applications such as home security, cleaning, etc. The robots need to add or update services frequently according to the changing needs of human users. Furthermore, reactive nature of the robots add complexity to develop robot applications. These challenges raise safety issues seriously. Considering that safe operation of home service robots is crucial, current practice of validating robot applications is, however, not mature enough for wide deployment of home service robots. In this paper, we present our experience of developing and formally verifying discrete control software of Samsung Home Robot (SHR) using Esterel. We give a brief background on Esterel, then illuminate our result in formally verifying stopping behavior of SHR. Through the verification, we could detect and solve a feature interaction problem which caused the robot not to stop when a user commanded the robot to stop.

Index Terms—robot programming, formal verification, discrete controller synthesis

I. INTRODUCTION

Accompanying advances in robotics, control theory, and computer science, personal robots (e.g. health care and home security, etc) have received strong support from both academia as well as industry. Home service robots have immediate impacts on increasing quality of human life in a wide range of potential applications.

Home service robots are required to achieve task goals in various situations. For example, during navigation, a robot must recognize obstacles that appear suddenly (e.g. pets or babies) and modify its path without damaging the obstacles. In addition, interaction between a user and a robot should be user-oriented, i.e., a robot should be intelligent enough to recognize various forms of commands such as voice and motion and generate output in a convenient way to the user. Furthermore, service features are dynamically changing to address the needs of the application areas, some of which are still being developed. Finally, many applications should satisfy stringent safety requirements.

Due to the above complicated challenges, the robot applications become increasingly complex. Thus, the applications and services are created by integrating technology intensive components (vision, speech, path planning, etc). Due to limited development resources, robot developers, however, tend to concentrate on technology oriented components at the early stage of product development without considering how they will be integrated. Consequently, initial products are often developed by integrating these components in an ad-hoc way, which often creates feature interaction problems [1] [2]. Feature interaction problems are hard to solve because it is difficult to see how components behaviors are coordinated by traditional validation methods such as code inspection or testing. Code inspection is effective to find static bugs such as missing initialization or type mismatches, but weak at detecting faulty behaviors. Testing has incomplete coverage and always leaves undetected bugs. Furthermore, building a test environment for robots is highly complicated, because reproducing the same fault is difficult due to nondeterministic behaviors of the robots caused by timing and concurrency. Consequently, testing and debugging processes often take more than half of total development time.

Therefore, it is a challenging task to validate and verify quality requirements such as safety and real-time properties of home service robots [3] [4] [5]. This task is, however, a prerequisite for wide deployment of home service robots. Necessity of formal validation and verification (V&V) has been well-recognized in robotics areas [6] [7]. Also, robot domain specific V&V frameworks such as ORCAAD [8] and MAESTRO [9] have been developed. In robot industry, however, a practice of applying formal V&V is not popular because there are not enough field experiences of formal V&V in the industry yet. We believe, however, that formal V&V can be a complementary solution for increasing quality of products.

In this paper, we describe our experience in formally verifying Samsung Home Robot (SHR) developed by Samsung Advanced Institute of Technology (SAIT). We re-engineered a discrete controller of SHR using Esterel [10] and performed formal V&V about stopping behaviors of SHR. After explaining brief background of the Esterel framework, we illuminate our verification results on SHR and describe a feature interaction problem detected and solved during the verification process.
II. BACKGROUND OF SHR

SHR is a prototype of home service robot for daily home services such as vacuum cleaning and controlling home appliances, etc. HW components and services of SHR are described in Sec II-B and Sec II-C respectively.

A. Project Background

SHR100 is a successor of SHR50 and SHR00. Development of SHR00 started in 2002 by four separate teams of SAIT consisting of 13 people working on speech recognition, vision recognition, map building, and actuator control. SHR50 as well as SHR00, however, exhibited often unstable behaviors such as missing user commands and stuttered movement although each part had worked successfully when not integrated (this kind of failure is not uncommon in robotics field [3]). As a consequence, SAIT decided to give up SHR50 and SHR00 and develop SHR100 from scratch. To prevent similar problems, SAIT requested POSTECH to design a software architecture after ten months into the new development (at that point, high-level specifications of SHR100 was documented. Also a part of control software was implemented including “call and come” service (see Sec II-C) and related features).

At that request, POSTECH reviewed the specifications and the implementation of SHR100, and then re-engineered existing implementation. POSTECH focused on producing working code of high reliability. With conventional programming framework using C/C++, it seemed difficult to achieve this goal. We decided to use the Esterel framework for its concise language for programming reactive systems and its support of formal verification by model checking. Furthermore, Esterel is a mature framework with commercial support [12].

B. Components of SHR100

SHR100 has a single board computer (Pentium IV 2.4Ghz with 512MB memory running embedded WindowsXP) controlling peripherals as follows.

- **Input peripherals**
  - 1 ceiling camera for building a map (640x480 resolution and 5 frames/s)
  - 1 front camera for recognizing users and remote surveillance (320x240 resolution and 15 frames/s)
  - 8 microphones for speaker localization and speech recognition (8 Khz sampling rate)
  - 1 structure light sensor for obstacle detection

- **Output peripherals**
  - 1 LCD display for information display
  - 1 speaker for speech generation
  - 2 actuators for right and left wheels

- **Input/output peripheral**
  - Wireless LAN for communicating to a home server

The components of SHR100 are illustrated in Figure 1.

C. Services of SHR100

Some of the primary services of SHR100 are described as follows.

- **Call and Come (CC)**
  This service first analyzes audio data sampled from eight microphones attached to the surface of the robot, to detect predefined sound patterns (e.g., hand clap or voice command). There are two commands “come” and “stop”. Once a “come” command is recognized, the robot tries to detect the direction of sound source by comparing the strength of sound captured by the eight microphones. Then, the robot rotates to the direction of sound source and tries to recognize a human face by analyzing video data captured by the front camera. If the caller’s face is detected, the robot moves forward until it reaches within 1 meter from the caller. A “Stop” command makes the robot stop. If command recognition, sound source detection, or face recognition fails, CC resets to the initial state. CC is preemptible, i.e., while CC is executed, newly recognized command makes the robot ignore a previous command and follow the new one.

- **User Following (UF)**
  This service is triggered right after CC is completed. The robot uses two inputs to locate the user: the front camera and the structured light sensor. Once UF is triggered, the robot constantly checks vision data from the front camera and sensor data from the structured light for locating the user. At the same time, the robot keeps following the user within 1 meter range. If the robot misses a user, the robot notifies user by speaking “I lost you” and UF turns into CC. Then, the user makes a “come” command to let the robot recognize the user and restart UF. Similar to CC, UF is a preemptible service.

- **Security Monitoring (SM)**
  The robot patrols around a house for surveillance using the map generated by Simultaneous Localization and Map building (SLAM) module. Intrusion or accidents are defined as patterns recognizable from vision and sound data. For example, intrusion can be detected by watching images and sounds from doors and windows. Once such an event is detected,
the robot notifies the user directly via an alarm or indirectly through a home server.

- Tele-presence (TP)
  A remote user can control the robot using a PDA. The robot sends the remote user a map of the house generated by the SLAM module periodically. The user can command the robot to move to a specific position in the map displayed at the PDA. In addition, the robot can send images obtained from the front camera to the remote PDA for surveillance.

III. BACKGROUND OF THE ESTEREL FRAMEWORK

A. The Esterel Language

Esterel is a programming language for reactive systems that wait for a set of inputs, and react to the inputs by computing and producing outputs, and then wait for new inputs again. Since Esterel is based on the “synchrony hypothesis” [10], every reaction to a set of inputs should be instantaneous. In practice, this means that a system should react to input signals before input signals of the next cycle arrive. Synchrony hypothesis considerably simplifies the specifications of reactive systems. Furthermore, many application areas satisfy this hypothesis.

A program written in Esterel specifies components (called modules) running in parallel. Modules communicate with each other and the outside world through input/output signals. Signals are broadcasted and may carry values of arbitrary types. Operators in the Esterel language have precisely defined mathematical semantics. An Esterel program has its “meaning” as a finite state Mealy machine (FSM) whose transition edges are labeled with pairs of input and output signals. The Esterel compiler automatically performs the interleaving between parallel modules and generates a single FSM in C regardless of numbers of parallel modules. Thus, the parallelism in Esterel is a structuring tool for programming convenience. For example, a program ABRO [13] emits an output O as soon as two inputs A and B have occurred. The ABRO program resets this behavior each time an input R occurs. Fig 2 shows the ABRO program and the corresponding FSM.

![Fig. 2. An ABRO example and the corresponding FSM](image)

B. The Esterel Toolset

Esterel toolset consists of three components: the esterel Esterel-to-C compiler, the xes graphical simulator, and the xeve [14] model checker.

1) The esterel Compiler: One advantage of Esterel over other formal modeling languages such as CCS or PROMELA is efficient C code generation. Once a developer has verified the correctness of an Esterel program, he/she can automatically generate correct C code without manual conversion from a formal specification to working code. Furthermore, esterel generates platform neutral C code so that a developer can port an Esterel program to different OS/HW platforms (e.g. Windows to Linux, or vice versa) without difficulty. Finally, esterel compiler provides interfaces between Esterel and C so that an Esterel program can call external C functions and existing C code can emit/receives signals to/from the Esterel program seamlessly.

2) The xes Graphical Simulator: xes supports interactive simulation as well as session recording/replay. Given an Esterel program, a user can execute the program symbolically selecting input signals to emit and advancing ticks (time instants) (see Fig 3). Thus, without detailed C implementation for hardware, software controller specified in Esterel can be simulated and tested. xes is also used to examine the execution trace of a counter example generated from xeve.

3) The xeve Model Checker: xeve minimizes and analyzes a FSM generated from an Esterel program. Basic verification process of xeve is to check presence of an output signal with given configuration of input signals by model checking [15]. First, a user selects input signals as “always present”, “always absent”, or “randomly present”. Then, the user selects an output signal to check if it can be emitted with given configuration of the input signals. Simple properties such as “if a user does not give a command to a robot, the robot must not move” (see Sec IV-B) can be checked simply in this way. More complex property can be checked by building an observer module which emits a violation signal when the property is violated (See Sec IV-C).

IV. FORMAL VERIFICATION OF SHR100

In this section, we describe safety properties $P_1$, $P_2$, and $P_3$ regarding stopping behavior of SHR100. We describe
the most primitive safety property $P_1$ first, then incrementally refine $P_1$ into $P_2$ and $P_3$. Sec IV-B and Sec IV-C show if the controller running CC only satisfies $P_1$, $P_2$, and $P_3$. Sec IV-D shows if the controller running CC with UF together satisfies $P_1$, $P_2$, and $P_3$.

A. Overview of the CC Implementation

We re-implemented a discrete controller of SHR100 written in C++ into Esterel after separating control oriented parts from data oriented parts (e.g., vision and speech recognition). In this section, we describe the CC service implementation in the control software as an example. A main control procedure for the CC service was implemented in ProcessState() illustrated in Fig. 4.

```c++
void processState() {
    switch(m_order) {
        case 0: STOP();
        case 1: ROTATE();
        case 2: static int nCount = 0;
        // ... 
        case 9: CNC_DONE();
        case 10: emit STOP; pause;
        // ... 
    }
}
```

Fig. 4. A main control procedure for CC service in C++

ProcessState() is called periodically once in every 100 milliseconds to react an input from a user. Given a user command, CC executes through sequential “steps”. Each step is identified by the value of $m_{order}$ and represented by corresponding case statements block. At the end of each case statements block, $m_{order}$ is updated to indicate the next step. After one step is executed, ProcessState() is terminated and called again after 100 milliseconds. If a new command is given between these two adjacent invocations, a previous command is ignored and the new command is processed.

This pattern of reactive programming is an straightforward way of implementing preemtion, but error prone. For example, at line 13, nCount is used for testing two times whether SHR stops rotation. May happen, however, only one time. This is because nCount is declared as a static local variable at line 10 and can be greater than two all the time. This error decreases accuracy of user recognition due to blurred image captured while the robot does not stop rotation completely. As more service features are added to SHR, the complexity of C/C++ code increases exponentially. Soon, a developer can hardly manage and debug the program.

Esterel prevents such errors by handling a preemptive event $e$ with a preemption operator EVERY $e$ DO statements END EVERY (see line 10 to line 21 in Fig 5). Fig 5 is a skeleton of the re-implemented CC service in Esterel. A module control_plane (line 1 to line 5) represents a whole control software including a CC service cnc, a UF service uf, and so on (see line 4). Communication among modules is implemented using input/output signals declared at line 2 and line 3. COME_COMMAND and STOP_COMMAND are input signals corresponding to the “come” and “stop” commands respectively. A “come” command is handled from line 12 to line 16 and a “stop” command is handled from line 17 to line 19. A task of rotating and recognizing the user is implemented as a submodule rot_det and executed at line 14. As we have seen, Esterel implementation defines components concretely using modules and shows interaction among the components clearly using explicit input/output signals. This feature facilitates testing and debugging interactions among components easily.

B. Verification of the CC Service without an Observer

Fig 6 shows the snapshot of xev, verifying the CC service only, not together with other services. Consider a following property $P_1$.

$P_1$: If a user does not give a command to the robot, the robot must not move.

There are only two input commands - COME_COMMAND and STOP_COMMAND. Also, there are only two output signals which can move the robot - GO and ROTATE. We can verify $P_1$ by setting COME_COMMAND and STOP_COMMAND as “always absent” (marked as blue in the left window of Fig 6), selecting GO as an output 0.

Notice that $P_1$ should be satisfied all the time and with all internal configurations/states of SHR100 which may change through executions (not with just one specific configuration/state).
signal to check, and pressing the “Apply” button of xeve. Then, xeve shows that GO is never emitted by the robot (indicated in the right window of Fig 6). In the same way, we can check whether ROTATE is emitted ever and see that ROTATE is never emitted. Thus, we can conclude that the CC service satisfies $P_1$.

Although this safety requirement $P_1$ looks obvious, this requirement is important for ensuring safe operation of the robot in a house. Violation of $P_1$ means that the robot can move autonomously without a user’s command and cause damage to furniture or a human. Furthermore, guaranteeing satisfaction of $P_1$ is a challenging task without model checking because a developer has to find out all possible test cases manually and perform testing one-by-one without error.

Slightly more refined property regarding movements of the robot can be described as $P_2$.

\[ P_2 : \text{If a user does not give a “come” command, but may give a “stop” command to the robot, the robot does not move.} \]

We can verify that the CC service satisfies $P_2$.

C. Verification of the CC Service using an Observer

To verify more complicated properties, we need to build an observer which is an Esterel module to detect violations of the properties. Suppose a property $P_3$.

\[ P_3 : \text{If a user gives a “stop” command, the robot stops and does not move without any new command.} \]

We can incorporate an observer described in Fig 7 with a cc module in parallel. Programming an observer is more familiar to most robot developers than writing a temporal logic (TL) [16] formula.\(^4\)

\[
\begin{align*}
01: & \text{module observer; } \\
02: & \text{input STOP\_COMMAND, COME\_COMMAND, ROTATE, STOP, GO; } \\
03: & \text{output STOP\_VIOLATION; } \\
04: & \text{weak abort } \\
05: & \text{every immediate STOP\_COMMAND do } \\
06: & \text{present STOP then } \\
07: & \text{loop } \\
08: & \text{present [ROTATE or GO]} \\
09: & \text{then emit STOP\_VIOLATION; } \\
10: & \text{end present; } \\
11: & \text{pause; } \\
12: & \text{end loop; } \\
13: & \text{end present } \\
14: & \text{emit STOP\_VIOLATION; } \\
15: & \text{end every } \\
16: & \text{when COME\_COMMAND; } \\
17: & \text{end module }
\end{align*}
\]

Fig. 7. An observer for detecting STOP\_VIOLATION

observer emits STOP\_VIOLATION at line 9 and line 14 if $P_3$ is violated. If a “stop” command is given (line 5) and the robot stops immediately (line 6), then the observer keeps watching if the robot rotates or moves forward (line 7 to line 12) unless any new command is given by the user.\(^4\)

We can see that STOP\_VIOLATION is never emitted with all possible configurations of input signals through xeve.

D. Verification of the Concurrent CC and UF Services

We checked if the control software consisting of the CC and UF services satisfied $P_1$ and $P_2$. We could see that the control software satisfied $P_1$, but surprisingly not $P_2$. Verification result said that ROTATE and GO could be possibly emitted when COME\_COMMAND command was absent and STOP\_COMMAND command might be given. In general, verification result of xeve is sound but not complete because a FSM is generated from an Esterel program without evaluating expressions. Therefore, a user should check whether a violation report is a real one or a false alarm. To support this activity, xeve has a facility of generating an execution trace of a counter example which can be simulated by xeve.\(^5\)

Through simulations of the Esterel program, we could figure out that UF made the robot rotate and move forward when a “stop” command was given (i.e., the violation was a real one). This was because that UF was triggered by CNC\_DONE which was emitted by CC when a “come” command or a “stop” command was successfully processed (see Sec II-C and line 16 and line 19 of Fig 5). UF should had been triggered only after a “come” command was processed, not after a “stop” command was processed. We refined CNC\_DONE into CNC\_COME\_DONE and CNC\_STOP\_DONE. Then, we modified the UF implementation so that only CNC\_COME\_DONE could invoke UF. After this modification, we could see that $P_2$ was satisfied by the concurrent CC and UF services.

We checked if $P_3$ was satisfied by the revised control software which had CC and UF running concurrently. We built an Esterel program having a UF module uf, a CC module cc, and an observer observer in parallel. We could use observer developed to check $P_3$ with the CC service only (see Fig 7) without modification. This was because change of the target system (uf being added) was not relevant to observer as long as interface between the target system and observer was identical. We could see that the control software satisfied $P_3$.

E. Experimental Result in the Verification

We used a WindowsXP machine with Pentium IV 2.8C and 1GB memory for the verification. Verification of $P_1$, $P_2$, and $P_3$ took less than 10 seconds and 128 MB memory, which was not burdensome to developers. Notice that what we had worked on was real implementation, not an abstract model. We replaced existing C/C++ implementation of control software loaded on the SHR100 hardware with C code generated from the Esterel program. By running the control software proved correct with regard to $P_1$, $P_2$, and $P_3$, SHR100 could operate with high reliability.

As we have seen through Sec IV-B to Sec IV-D, defining safety properties rigorously takes considerable effort. Such

\(^4\)We can translate a safety property written in temporal logic into an observer in Esterel [17].

\(^5\)xeve v5.92 on Windows platform which we used, however, had a bug that an execution trace of a counter example was not correctly generated.
effort can, however, reduce overall development cost and field operation cost by increasing reliability of applications and reducing testing and debugging processes.

V. CONCLUSION

We have reported a case study of developing and verifying discrete controller of a home service robot SHR100. Given various service requirements of home service robots, coordinating diverse services in timely and safe manner is a significant challenge. The gist of our approach is to develop and verify systems in an integrated framework, not separately, to increase reliability of SHR100. We used Esterel to develop working code of SHR100 and to verify safety requirements on stopping behaviors. We demonstrated that formal V&V could increase the reliability of a working system; using model checking, we detected and solved a feature interaction problem which caused the robot not to stop when a user gave a “stop” command. We believe that the cost of learning and adopting a new programming language like Esterel can be paid back by increased product quality as well as reduced testing/debugging time.

We will handle the resource management problem which is frequent source of unstable behaviors such as stuttering movement and ignorance of user input under heavy resource utilization. For this purpose, Monitoring and Checking framework [18] can be explored as an additional help.

REFERENCES


