Using FRP in Yampa to Redesign the Control Software for the Robotic Artwork “Pygmies”

Student

Eliane Irène Schmidli

Definition of Task: The Pygmies artwork by Pors & Rao (see Figure 3) is a robot application reacting to the sounds in the room and controlling actuators. The control software is written in a low-level imperative style. As a result, the program sequence and the commands to the actuators are intertwined which makes the code difficult to understand. Furthermore, the program code describes the behavior of the program in certain states, but it is difficult to comprehend where the state transitions are initiated. This makes changes in the program sequence difficult.

Functional Reactive Programming (FRP) is a composable, modular way to program reactive applications. With FRP, the control software has been redesigned, making it more customizable, especially for people with little programming experience. To see with little effort what such a new design would look like, a graphical user interface (GUI) application was developed that simulates the artwork. The redesign uses Yampa, an FRP implementation in Haskell.

Approach: In FRP, it is possible to separate the behavior of the program and the control of the sensors and actuators. The behavior is implemented in a function 'pygmy' consuming a signal indicating the current sound level and producing a signal indicating the current position of a figure. The GUI can use the resulting signal to draw a figure at the corresponding position.

A program in an imperative style describes what should happen in a certain state (see Figure 1). But, it is unclear when the Pygmy instance is set to the corresponding state. In FRP, the reaction to events is described (see Figure 2). For example, in the code from lines 9 to 10, the 'safeBehavior' is executed until an event called 'danger' occurs, after which the Pygmy will hide. Or from lines 14 to 15, the Pygmy will wait until the time is up and then move until it arrives at a certain position.

Conclusion: The implementation of the control software of the Pygmies artwork with FRP is very promising. Due to the modularization and the more visible state transitions, the code is better understandable, and thus it is easier to make changes in the code.

The implementation in FRP comes close to the original implementation. Due to the many advantages that the new design in FRP brings, it is worth replacing the control software of the artwork. Thanks to the separation of the GUI component, it should be possible to replace the GUI with the control of the actuators without adapting the behavior of the Pygmies much.

Figure 1: Old implementation of the behavior of the Pygmy using imperative style (simplified)
Own presentment

```
1  def program(state):
2    switch state
3
4      case 'hide':
5         actuator.move(0, hidingV)
6            state = 'go_standing'
7            waitUntil = currentTime + waitingTime
8
9      case 'go_standing':
10        if currentTime > waitUntil:
11           actuator.move(standPos, standV)
12           state = 'wait'
13
14     case 'wait':
15        # do nothing
16        pass
```

Figure 2: New implementation of the behavior of the Pygmy using FRP (simplified)
Own presentment

```
1  pygmy :: SF Sound Position
2  pygmy = proc sound -> do
3      rec
4          v <- program -< (pos, sound)
5          pos <- integral -< v
6      returnA -< pos
7
8  program :: SF (Sound, Pos) Vel
9  program = (safeBehavior &&& danger)
10     `switch` \_ -> (goHiding &&& arrival)
11     `switch` const program
12
13 safeBehavior :: SF (Sound, Pos) Vel
14 safeBehavior = (wait &&& timeIsUp)
15     `switch` \_ -> (goStanding &&& arrival)
16     `switch` const wait
```

Figure 3: Pygmies artwork by Pors & Rao
Source: http://www.porsandrao.com/work/?workid=21

Advisor
Prof. Dr. Farhad D. Mehta

Subject Area
Computer Science

Project Partner
Pors & Rao, Bangalore, India