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

Follow-up

Student

Eliane Irène Schmidli

Initial Situation: The Pygmies artwork by Pors & Rao (see Fig. 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. Last semester, the control software has been redesigned with FRP, and a graphical user interface (GUI) application was developed that simulates the artwork. The redesign uses Yampa, an FRP implementation in Haskell. This semester the redesign was further developed and connected to the hardware of the artwork.

Approach: The code of the redesign was revised so that it is more understandable. In contrast to the imperative style (see Fig. 1), the redesign clearly shows when the state transitions happen (see Fig. 2). For example, in the code from lines 7 to 8, the 'creepOut' is executed until an event called 'danger' occurs, after which the Pygmy will hide.

The artwork's peripherals are controlled using a Python framework. Instead of processing the microphone input itself, the framework now sends it to the Haskell service via a socket connection. This service uses FRP to produce a signal that displays the current positions of the pygmies during runtime. The current values are sent to the Python framework in short intervals. The framework then moves the actuators to the corresponding positions. This way, the behavior of the pygmies is decoupled from the control of the sensors and actuators.

The new software was tested with a variation of the Pygmies artwork called "Lone Pygmy". In this artwork, there is one Pygmy on each side of the panel. Only one of the four Pygmies peeks out at a time. As soon as it hides, the side is changed. This gives the impression that it is always the same Pygmy. The implementation of this variant required only a few adjustments. Instead of calculating several positions, only the behavior of one Pygmy is executed. The remaining three positions are set to 0. As soon as the Pygmy hides, the position of the Pygmy is randomly swapped with another position.

Conclusion: The modular structure of the FRP code makes it possible to make changes to the behavior (e.g. implement Lone Pygmy) without adapting the hardware control.

The produced position signal can be consumed by any output device. Therefore, it was easy to replace the GUI with the actuator control. It is now possible to use both variants and to test a change in the program first in the GUI before it is executed on the hardware.

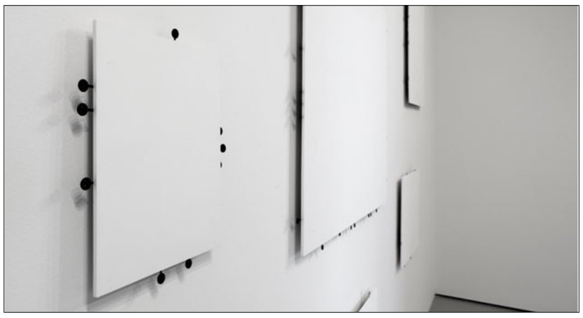
Figure 1: Old implementation of the behavior of the Pygmy using imperative style (simplified)
Own presentment

```
def program(state):
    switch state

        case 'hide':
            actuator.move(0, hidingV)
            waitUntil = currentTime + waitingTime
            state = 'go_standing'

        case 'go_standing':
            if currentTime > waitUntil:
            actuator.move(standPos, standV)
            state = 'wait'

        case 'wait':
            # do nothing
            pass
```

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

```
 1  creepOut :: SF (State, Pos, Sound) Vel
 2  creepOut = (stop `doUntil` timeToStand)
 3     `switch` \mov -> (goStanding mov)
 4     `switch` const stop
 5
 6  behavior :: SF (State, Pos, Sound) Vel
 7  behavior = (creepOut `doUntil` danger)
 8     `switch` \mov -> (goHiding mov)
 9     `switch` const behavior
10
11  pygmy :: SF (PygmyState, Sound) Pos
12  pygmy = proc (p, sound) -> do
13   rec
14     v <- program -< (p, pos, sound)
15     pos <- integral -< v
16    returnA -< pos
```

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