Teaching Students of Engineering some Insights of the Internet of Things using Racket and the RaspberryPi

Daniel Brunner
Systemhaus Brunner & Brunner Software
Schulstr. 8
Biedenkopf 35216, Germany
daniel@dbrunner.de

Stephan Brunner
Systemhaus Brunner & Brunner Software
Schulstr. 8
Biedenkopf 35216, Germany
stephan.brunner@systemhaus-brunner.de

ABSTRACT

We gave a course to teach students of engineering some insights into the Internet of Things (IoT). We started with an introduction into programming using Racket and the *Beginner Student Language (BSL)* teachpack. After that we introduced the RaspberryPi[1] and showed how to read data from a thermal sensor and switch a LED. With this knowledge we taught students to implement a simple publish-subscribe pattern where the RaspberryPi collected some thermal data and a program on their PC monitored these data and could switch the LED depending on the measured temperature. With this setup we explained several aspects of distributed computing and the Internet of Things in particular.

CCS CONCEPTS

•Social and professional topics →Computing education; •Applied computing →*Engineering;* •Computer systems organization →*Sensors and actuators;*

KEYWORDS

Racket, BSL, RaspberryPi, IoT, distributed computing

ACM Reference format:
Daniel Brunner and Stephan Brunner. 2016. Teaching Students of Engineering some Insights of the Internet of Things using Racket and the RaspberryPi. In *Proceedings of 10th European Lisp Symposium, Brussels, Belgium, April 3–4, 2017 (10th ELS),* 2 pages.
DOI:

1 INTRODUCTION AND GOAL

Nowadays a lot of news is written about the "Internet of Things" (IoT). In Germany these news are often related to the so-called "Industrie 4.0", a term which describes a new way to organize processes of manufacturing. Although the term is often used in newspapers and even at our universities, a precise and widely accepted definition does not yet exist. But most authors would agree that digitalization influences manufacturing and leads to new forms of manufacturing or even to additional product-related services. Therefore we took this as a starting point and our goal was to teach students of engineering some basic principles on how to design a distributed application.

The basic idea was to have a small device (RaspberryPi with a thermal sensor) which monitors some state of an imaginary machine and sends this data to a central message broker. A separate monitoring system (on the student's PC) should subscribe to these messages and check if the monitored device is in some "healthy" state. If the measured data was out of a given range, some action should be initiated (switch on a LED which was connected to the RaspberryPi).

Although there are a lot of IoT suites available, we wanted to accomplish this setup with very basic tools. On the other hand we wanted to avoid diving into assembler programming or system programming with C. Therefore we chose Racket along with the Beginner Student Language and the *universe* teachpack from the "How to Design Programs" (HtDP) teaching materials.

With these in hand we gave a two-day course at the dual study program *StudiumPlus*[2] of the University of Applied Sciences of Central Hesse (Technische Hochschule Mittelhessen[3]). Most of our students only had little knowledge in programming. Some of them took a C++ course at the beginning of their studies. Therefore we could not rely on any programming skills.

2 BASIC PROGRAMMING

To teach some basic ideas of programming we chose the approach of "How to Design Programs (2nd edition)" (see [2]). We taught some of the material of the first chapters and emphasized on the structural design recipe (see [1]). Students could follow and perform their exercises with Racket's IDE, DrRacket, which works on the student's PC as well as on the RaspberryPi. This took about one third of the whole course.

We tried to limit the material to what really is essential to build a small distributed application. Therefore we taught students about some basic data type of *BSL*: images, numbers, strings, booleans as well as some functions on how to work with these data types. After introducing booleans we went on to teach some logical operators and ended up with conditionals (`cond` and `if` and some predicates). These concepts were introduced using the *universe* teachpack which

[1]http://www.raspberrypi.org

[2]http://www.studiumplus.de
[3]http://www.thm.de

implements interactive, graphical programs (so-called *world* programs). Along with the data types we introduced the definition of functions and constants using `define`, comments, defining and using structs and how to use `require`.

3 SETUP

After teaching some basics in programming we set up the following components to establish a simple publish-subscribe pattern using the *universe* teachpack, e.g. connecting the *world* programs to the *universe server*, a central control program. To keep things very simple we omitted any authentication, encryption etc. We built teams consisting of two students. Each team received a RaspberryPi.

3.1 Student's PC and RaspberryPi

We provided the students with two modules: one with some basic struct definitions and constants to establish the publish-subscribe pattern and a second one where we hid some system calls to obtain the temperature using the I^2C bus or switching the LED via the GPIO. This was done by simple calls to the command line tools `i2cget` and `gpio`.

We prepared the RaspberryPi with a thermal sensor (LM75, see [3]), a red LED and the newest Raspbian[4] image which contains a graphical user interface. On top of that we installed DrRacket. Therefore students could use a simple VNC viewer to connect to the RaspberryPi and develop their programs directly on the RaspberryPi using the same IDE. Consequently students did not have to bother with Linux' command line programs.

Alternatively they could use the VNC viewer's function to upload files. Furthermore having a GUI is a requirement for the *universe* teachpack because the *world* programs need a `to-draw` clause.

The task for the student's PC was to develop a program which subscribed to the messages of their RaspberryPi, check if it is in a given range of temperatures and publish a suitable "change state" message.

After setting up the PC, the RaspberryPi and the above-described programs the students should monitor their sensor's temperature and watch for the LED to light up which it would if the temperature was out of a given range (e.g. using some cooler spray on the sensor).

3.2 Instructor's PC

The instructor's PC was running the message broker, a simple *server* program that implemented the described protocol. The source code of this component was not shown to the students. They were only taught how to publish and subscribe to data. To give a good overview over all RaspberryPis and their states, we built a small program that subscribed to all messages from the RaspberryPis and showed their states to the whole class via projector.

[4]http://www.raspbian.org

4 DISCUSSION

After the setup was working, we could explain several aspects of distributed systems and IoT in particular like the availability and type of network, low latency, bandwith restrictions, network congestion, authentication, message signing as well as topics like data format of the sensor, issues with time zones, different protocols and file formats. We ended up giving some advice on how to debug such distributed systems if something goes wrong.

Using Racket and the *universe* teachpack (and omitting lots of security measures) resulted in very short programs: Our sample implementation for student's components consisted of 67 lines for the program on the RaspberryPi and 95 lines for monitoring the messages. The modules that hid some implementation details on the structs and system programming (LM75, LED) used another 176 lines. Therefore students were not distracted by an overhead caused by libraries or the programming language itself. On the instructor's side more code was necessary: the message broker took 321 lines and our "overview" another 107 lines. To sum up: although only a fraction of Racket's BSL plus the *universe* teachpack was used, we were able to come up with a very short working solution.

After the two-day course students reported that they learned very fast and were suprised, they could achieve some results with only little knowledge in programming. We hoped that omitting authentication and encryption would encourage some of the students to hack or at least play with some of the other RaspberryPis. But that did not happen. Maybe they should be given more time to experiment and to think about possible shortcomings.

In a future course we are going to spend more time on the aspects of distributed computing, e.g. in terms of the *universe* teachpack: connecting *world* programs with a *universe*. This should broaden the understanding of the IoT and help the students to implement some of these techniques in their professional life.

ACKNOWLEDGMENTS

We would like to thank two anonymous reviewers for their helpful advice and Gerd Manthei of *StudiumPlus* at Technische Hochschule Mittelhessen who made this course possible.

REFERENCES

[1] Matthias Felleisen. 2015. Growing a Programmer. (8 September 2015). http://www.ccs.neu.edu/home/matthias/Thoughts/Growing_a_Programmer.html

[2] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. 2014. *How to Design Programs* (2nd ed.). MIT Press, Cambridge MA. http://www.ccs.neu.edu/home/matthias/HtDP2e/

[3] National Semiconductor 2001. *LM75: Digital Temperatur Sensor and Thermal watchdog with Two-Wire Interface.* National Semiconductor. http://esd.cs.ucr.edu/labs/temperature/LM75.pdf