Home Automation using Logic Programming as a Service

Kevin Michael Frick

Collegio Superiore dell'Università di Bologna

Contents

1 Abstract **1**

2 Manipulation of light switches **1**

2.1 RPi GPIO manipulation 2

2.2 Servomotors 2

2.3 Configurator Interface predicates 3

2.4 Client Interface predicates 3

3 User interface **3**

4 Final remarks **4**

1 Abstract

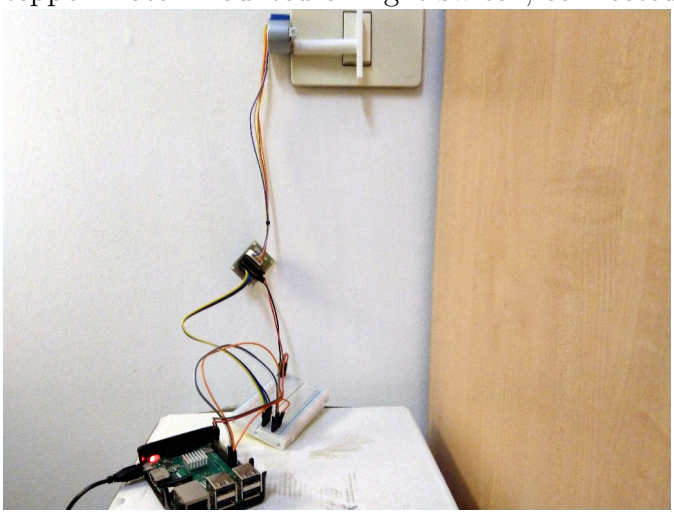
This paper introduces how a simple home automation system can be set up using the Logic Programming as a Service (LPaaS) [2] framework. First the system's internal workings are explained: light switches are opened and closed using a Raspberry Pi that controls stepper motors using its GPIO pins, which in turn are manipulated with tuProlog using its OO Library. Then the application's user interface, written using JavaFX, is shown and explained. Notable emphasis is given on the interaction between the Java VM and tuProlog back-end.

2 Manipulation of light switches

A LPaaS service is composed of a Configurator Interface, which requires authorization and allows to configure the service's KB, its admissible goals and - if the service allows stateful queries - its current state, and a Client Interface, which allows to query the

service for a solution. [2] In the case of light switches, the interaction can be stateful or stateless depending on the kind of switch that is installed: a toggle switch requires stateless interaction while an on-off switch requires stateful interaction. In both cases, however, the only command that can be given to a switch is the same: *toggle its state*. A stateless interaction merely presses the button, which takes care of inverting the light's state, while a stateful interaction turns the light on if it's off and vis versa. In any case, knowing whether the light is on or off requires stateful interaction, unless this information comes from something else, e.g. a light sensor.

Figure 1: Stepper motor mounted on light switch, connected to the RPi



2.1 RPi GPIO manipulation

By far, the most commonly used language in homebrew projects involving use of the RPi's GPIO interface is Python. Being an interpreted language, however, this introduces quite a bit of overhead - it's 300 times slower than direct manipulation using plain C[3] - and, most importantly, does not lend itself well to being integrated in a tuProlog environment. tuProlog's OOLibrary enables one to use Java inside Prolog while the its Java API allows using Prolog from Java, thus making it possible to seamlessly use JavaFX as a front-end for tuProlog which in turn manipulates the RPi's GPIO pins using a library provided by the Pi4J project [1].

2.2 Servomotors

The servomotor used for demonstration purposes is a 28BYJ-48 stepper motor with a ULN2003 driver. This stepper motor isn't able to produce enough torque to move some harder switches on the market, but it allows for easy and accurate control using a RPi's GPIO pins. A single step forward or backward is commanded by turning on and off the driver's four inputs in a certain sequence, detailed in the code. A single step equals 1/2048th of a full turn. The motor is implemented in Java as a `Stepper` class which pro-

vides the `turnOn()` and `turnOff()` public methods, which in turn call the `forward()` and `backward()` private methods which, using `setStep()`, change the state of the ULN2003's inputs in order to move the motor. Because of limitations of the driver, `setStep()` cannot be called more than once every ~1 ms.

2.3 Configurator Interface predicates

The only predicate in the Configurator Interface `set_is_stateful/2`. `set_is_stateful(Switch, IsStateful)` allows the user to configure each individual switch for stateful interaction (i.e. an on-off switch that needs to move the servomotor in two different ways depending on whether the switch is currently on or off) or stateless interaction (i.e. a toggle switch that is always pressed the same way). This in turn influences the behavior of the Client Interface predicate `toggle_state/1`. The current configuration is saved by modifying the theory at runtime using `assert/1` and `retract/1`.

2.4 Client Interface predicates

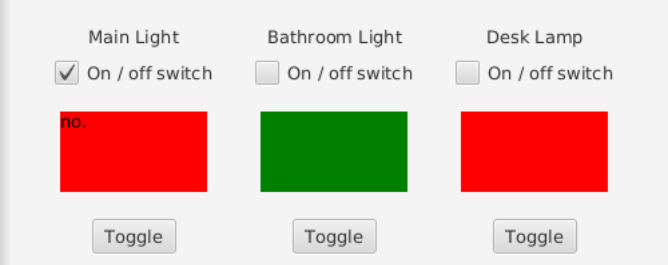
The application is structured into a traditional MVC pattern where tuProlog bridges the gap between the controller and the actual motion of the servomotor. The main predicate is `toggle_state/1`.

`toggle_state(Switch)` behaves in two different ways whether `stateful(Switch)` is true or false. If it's true it calls the `turnOn()` or `turnOff()` members of the `Stepper` Java class, depending on the atom `on(Switch)`, then inverts the truth value of said atom using `assert/1` or `retract/1`. If, instead, the switch is a toggle switch, it calls the `turnOn()` function of the corresponding `Stepper` and inverts the value of `on(Switch)`.

3 User interface

The user interface is written using JavaFX. A number of `Button`s equal to the lights that were fitted with the motors allow toggling their state, while an equal number of `TextFlow`s mark their state as given by tuProlog. A `CheckBox` for each switch allows it to be seen as a toggle switch - stateless interaction, default - or an on-off switch - stateful interaction. The `Button`s call the `toggle_state/1` predicate inside tuProlog, which toggles the light's state and changes the content of the corresponding `TextFlow` accordingly. The `CheckBox`es call the `set_is_stateful/2` predicate inside tuProlog.

Figure 2: User interface



This UI design is easily portable to a mobile environment such as an Android app.

4 Final remarks

Relevant code is available at `https://github.com/kmfrick/lpaas-home-automation`, hosted on GitHub. While the system presented in this paper is quite simple, it shows that a LPaaS system is easily integrated with IoT devices and lends itself to expansion: other than household lights, a similar system could control doors, blinds, smart appliances. The LPaaS framework using tuProlog allows seamless integration with the Java VM, thus making it possible to interface with devices commonly used in homebrew IoT projects, like RPis and Arduinos.

References

[1] Pi4J Project. `https://pi4j.com/1.2/index.html`.

[2] Roberta Calegari, Enrico Denti, Stefano Mariani, and Andrea Omicini. Logic programming as a service. *Theory and Practice of Logic Programming*, 18(5-6):846–873, 2018.

[3] Joonas Pihlajamaa. Benchmarking Raspberry Pi GPIO speed. `http://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/`.