ÉCOLE POLYTECH'NICE SOPHIA
SI5
2021-2022

Domain Specific Language

ArduinoML

Project delivery

Two different projects have to be delivered.
The first one should provide the basis arduinoML and (at least) one extension in both an internal and an external DSL. From the delivered DSL, it should be possible to write programs from which an arduino compliant code should be generated.
The second project will include the implementation in either an internal or an external DSL of another language that will be announced after the first delivery.
Deliveries are expected by email (to Julien Deantoni: firstname.name@univ-cotedazur.fr, with [DSL] as object prefix). First delivery is expected **before** 02.03.22, 10:00PM Paris Time and the second delivery before 03.01.22 10:00PM Paris Time. The delivery is expected as a pdf report (please, do not write a novel, but use few words in a clever, effective and scientific way!). The report must contain :

- the name of people in your teams
- a link to the code of your DSL(s)
- a description of the language(s) developed:
  - The domain model represented as a class diagram;
  - The concrete syntax represented in a BNF like form;
  - A description of your extension and how it was implemented;
- a set of relevant scenarios implemented by using your language(s) (internal and/or external);
- A critical analysis of (i) DSL implementation with respect to the ArduinoML use case and (ii) the technology you chose to achieve it;
- Responsibility of each member in the group with respect to the delivered project.

Objectives: Define the ArduinoML language

Your objective here is to enhance the kernel available in the ArduinoML zoo[1] with new features, and deliver a fully-fledged DSL that add value for the final users of ArduinoML. You are asked to develop two implementations of the very same language: one using an external approach (e.g., AntLR, Langium, Lex/Yacc, MPS, XText, Sirius, Racket) and the second one using an embedded one (Groovy, Ruby, Java, Scala, ...). On top of a common kernel, you are asked to introduce in your language one extension (among five) described at the end of this document.

[1] `https://github.com/mosser/ArduinoML-kernel`

Basic scenarios

1. **Very simple alarm**: Pushing a button activates a LED and a buzzer. Releasing the button switches the actuators off.
2. **Dual-check alarm**: It will trigger a buzzer if and only if two buttons are pushed at the very same time. Releasing at least one of the button stop the sound.
3. **State-based alarm**: Pushing the button once switch the system in a mode where the LED is switched on. Pushing it again switches it off.
4. **Multi-state alarm**: Pushing the button starts the buzz noise. Pushing it again stop the buzzer and switch the LED on. Pushing it again switch the LED off, and makes the system ready to make noise again after one push, and so on.

Commmon Parts

The following parts must be available in your DSL:

- **Abstract syntax**: The abstract syntax should be clearly identified in the delivered code.
- **Concrete syntax**: The concrete syntax (external or embedded) must be clearly identified and illustrated by a relevant set of scenarios. The syntax must leverage the tool chosen to implement it to make it clear and easy to use.
- **Validation**: Support your end-user by checking that a model is realizable on the Arduino platform. For example, compatibility of digital bricks with the pin used, termination of the modeled behavior, ...
- **Code generation**: Provide a generator producing turn-key arduino code, implementing the behavior. This code can be copy-pasted into the Arduino IDE and uploaded to a micro-controller.

"À la carte" features

The remainder of this document describes five extensions that you can implement on top of your kernel. Each extension is defined by a short description and at least one acceptance scenario. Choose (at least) one feature to introduce in your project.

Exception Throwing

To implement this extension, we assume that a red LED is always connected on a given port (e.g., D12). One can use ArduinoML to model erroneous situations (e.g., inconsistent data received, functional error) as special states. These error states are sinks, associated to a given numerical error code. When the sketch falls in such a state the red LED blinks conformingly to the associated error code to signal the error to the outside world. For example, in an "error 3" state, the LED will blink 3 times, then a void period, then 3 times again, etc.

> Acceptance Scenario: Consider a sketch with two buttons that must be used exclusively, for exemple in a double-door entrance system. If the two buttons are activated at the very same time, the red LED starts to blink and the sketch is blocked in an error state, as the double-door was violated.

Temporal transitions

ArduinoML does not support temporal transitions, i.e., transitions that are triggered a specific amount of time after entering in a state. This extension will allow it. Of course, such extension implies support for several output transitions from a single state.

> Acceptance scenario: Alan wants to define a state machine where LED1 is switched on after a push on button B1 and switched off 800ms after, waiting again for a new push on B1.

Supporting the LCD screen

The Electronic Bricks kit comes with an LCD screen. As an ArduinoML user, one can use the language to write text messages on the screen. These messages can be constants (e.g., "Hello, World!"), or built based on sensors or actuator status (e.g., "push button := ON", "temperature := 25 deg", "red light := ON"). One also expects the language to statically identify messages that cannot be displayed (e.g., too long). Moreover, using the bus connection prevent the use of several pins on the board (see the bus datasheet, on my website).

> Acceptance Scenario: The value of a specific sensor or actuator is displayed on the LCD screen as specified in the model.

Handling Analogical Bricks

The kernel only supports digital bricks. As a user, one can use the ArduinoML language to use analogical bricks. For example, the temperature sensor delivers the room temperature. Thus, analogical values can be exploited in transitions (e.g., if the room temperature is above 35 Celsius degrees, trigger an alarm). Analogical values can be set to analogical actuators (e.g., the buzzer or a LED), as constants or as values transferred from an analogical sensor.

> Acceptance Scenario: considering a temperature sensor, an alarm is triggered if the sensed temperature is greater than 57 Celsius degrees (fire detection).

Parallel periodic Region

ArduinoML supports the definition of a single state machine, so that it can be difficult to express orthogonal periodic behaviors; where the interleaving of all actions and timings make the state machine less manageable. This extension will allow the definition of various parallel state machines that run with a specific period. The generated code can then make use of multi tasking libraries like offered by Arduino (e.g., SimpleTimer (`https://playground.arduino.cc/Code/SimpleTimer` or Task scheduler (`https://playground.arduino.cc/Code/TaskScheduler`).

> Acceptance scenario: Bob wants to check the state of a button B1 every 400ms to toggle LED1, and he also wants to check the state of B2 every 50ms to switch on LED2 when pushed and switch it off when released.