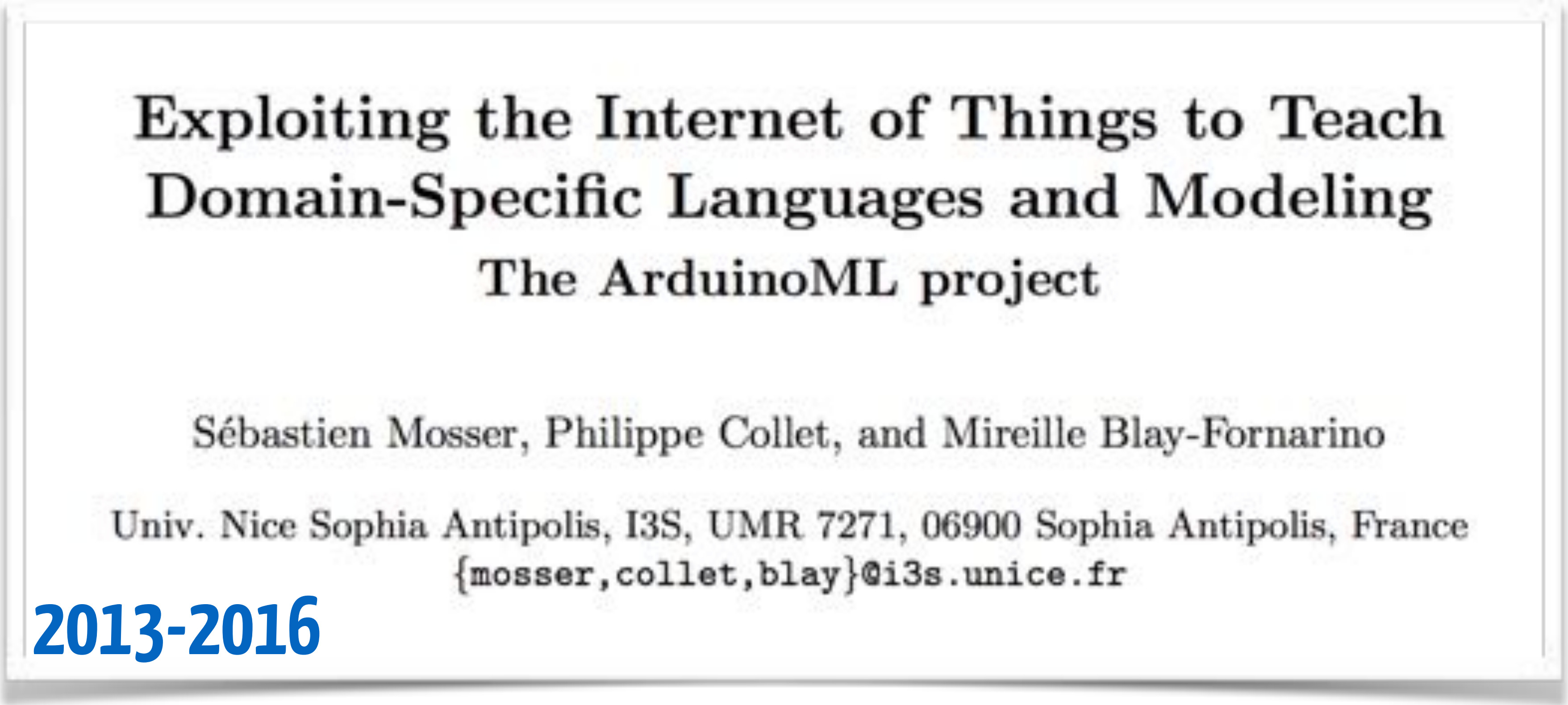
Practicing DSLs: From Code to Models

Laure Gonnord / Sébastien Mosser
Lyon 1, LIP / UCA, I3S
EduSump'18, 16.10.2018

pictures: sxc.hu

First version: EduSump'14



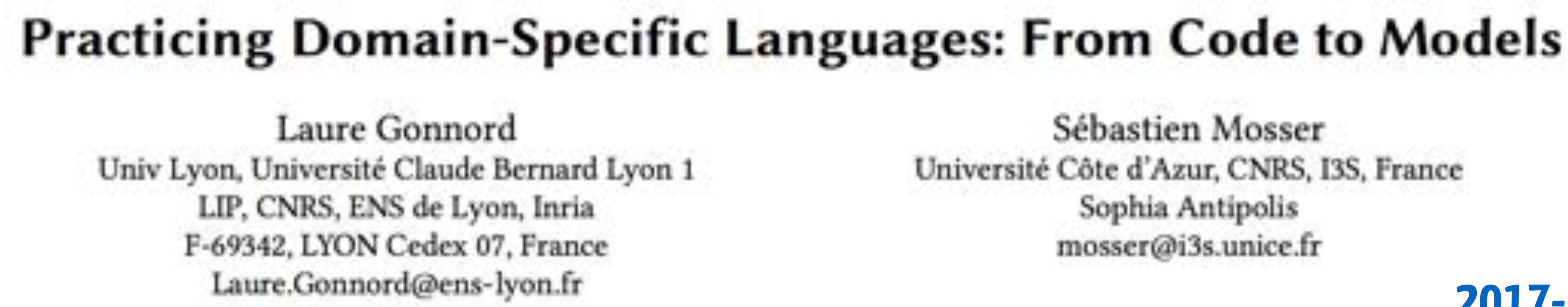
Exploiting the Internet of Things to Teach Domain-Specific Languages and Modeling

The ArduinoML project

Sébastien Mosser, Philippe Collet, and Mireille Blay-Fornarino

Univ. Nice Sophia Antipolis, I3S, UMR 7271, 06900 Sophia Antipolis, France
{mosser,collet,blay}@i3s.unice.fr

2013-2016



Practicing Domain-Specific Languages: From Code to Models

Laure Gonnord
Univ Lyon, Université Claude Bernard Lyon 1
LIP, CNRS, ENS de Lyon, Inria
F-69342, LYON Cedex 07, France
Laure.Gonnord@ens-lyon.fr

Sébastien Mosser
Université Côte d'Azur, CNRS, I3S, France
Sophia Antipolis
mosser@i3s.unice.fr

2017-...

Lyon (Compilation)

“Brainstorming”

Nice (MDE, DSL)

Terrible news!

Students consider Modelling as Meta-bla-bla-bla

Complicated tools

Compilation ≠ parsing

MDE is not “UML2RDMS”

Who cares about the UML in real life situations?

Terrible news!

Students consider Mod

ools

Co

MDE is not “UML2RDMS”

Wh about the UML in real life situations?

Blatant Advertisement:

Survey paper this afternoon!

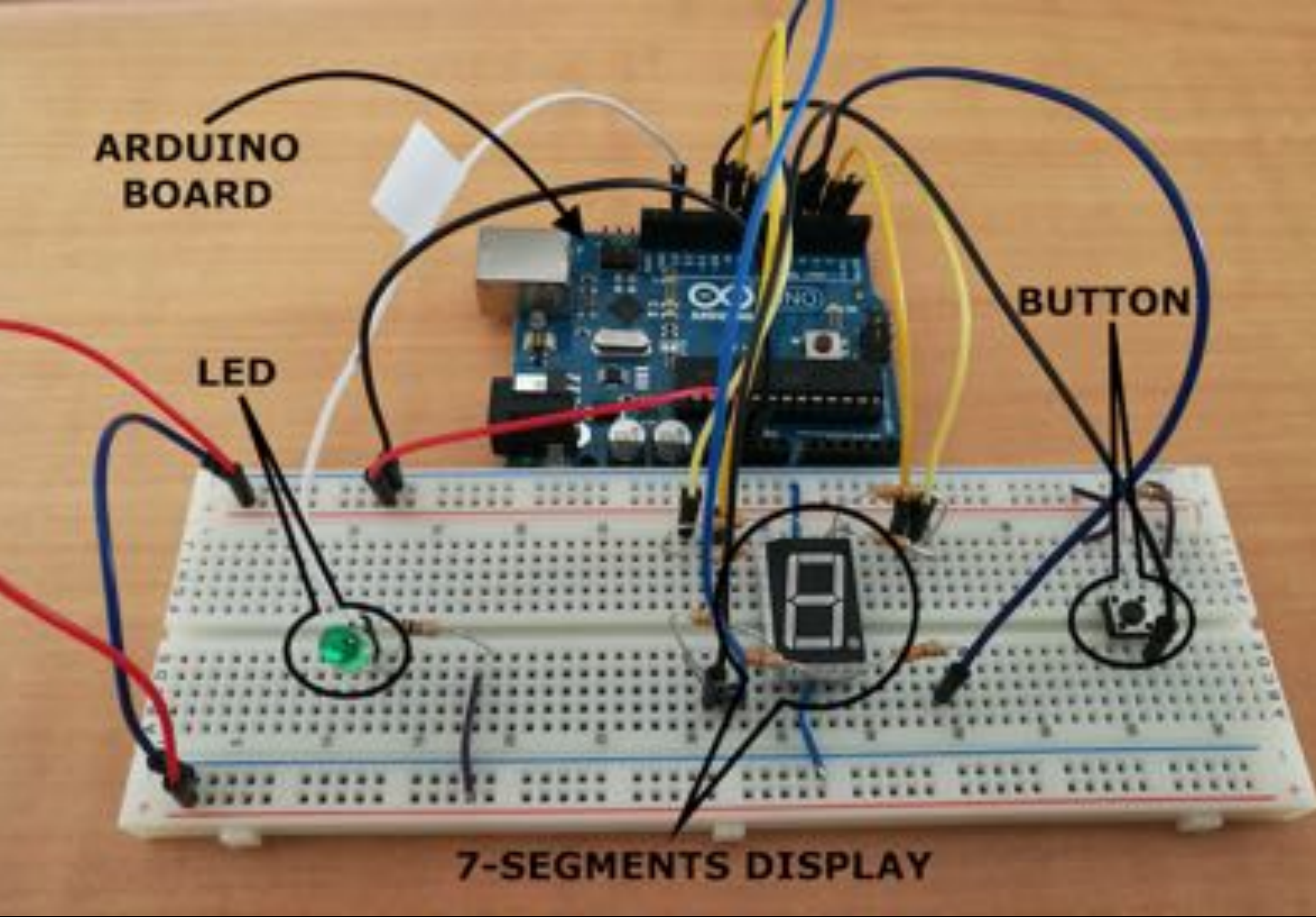
Objectives: Make modelling fun again!

1. Abstract code into models;
2. Operationalise model;
3. Concepts ≠ Tools;
4. Hands-on labs / projects.

Finding the Right Application Domain

- The domain cannot be a “toy example”
  - The domain should not be too much complicated
- The domain should illustrate nicely different modelling approaches
  - The domaine should also illustrate common challenges in compilation

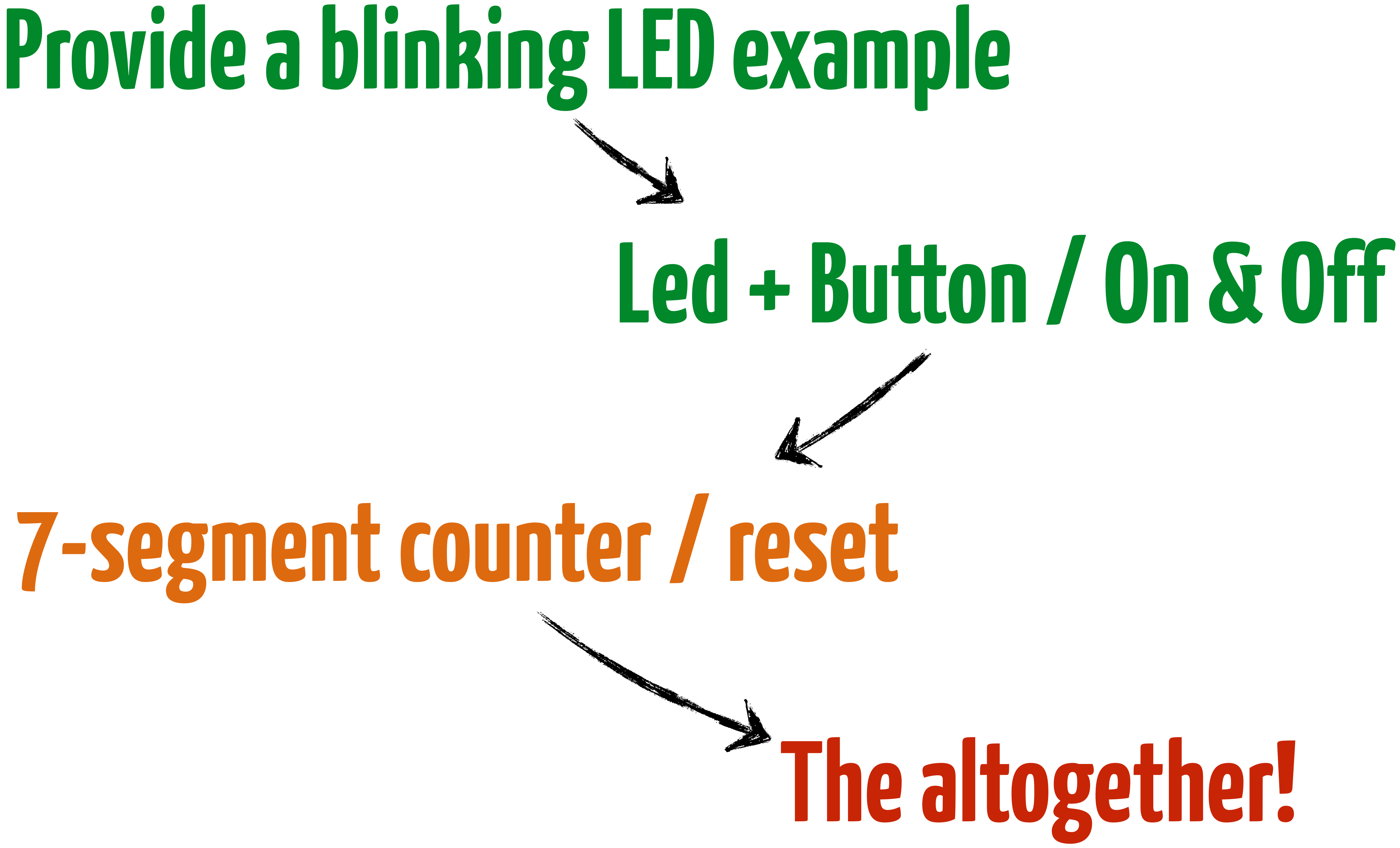
Reactive systems & Micro-controllers



Approximate cost = 30€

“Clean” version

Devil’s plan: Simple → Complicated



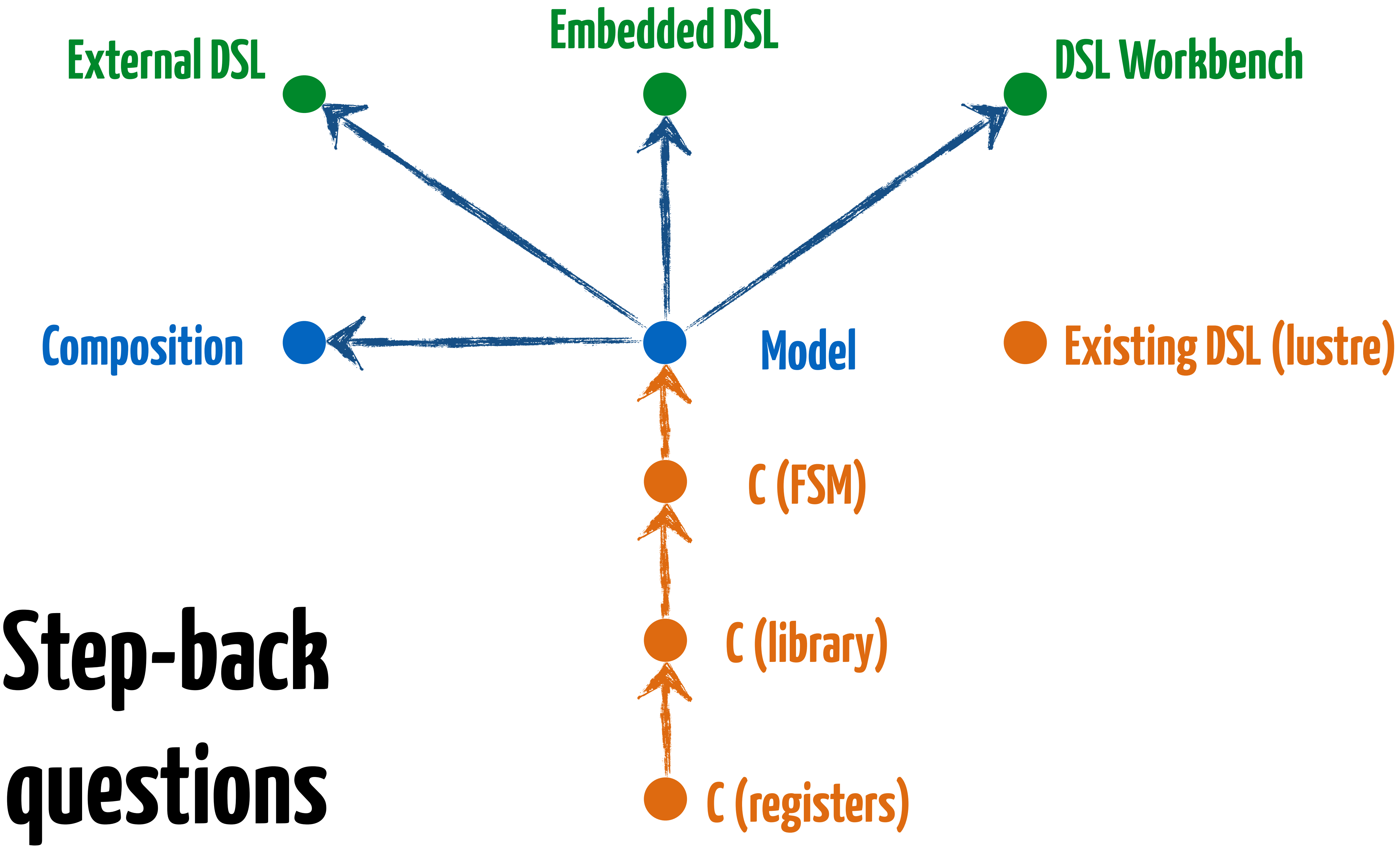
Provide a blinking LED example

Led + Button / On & Off

7-segment counter / reset

The altogether!

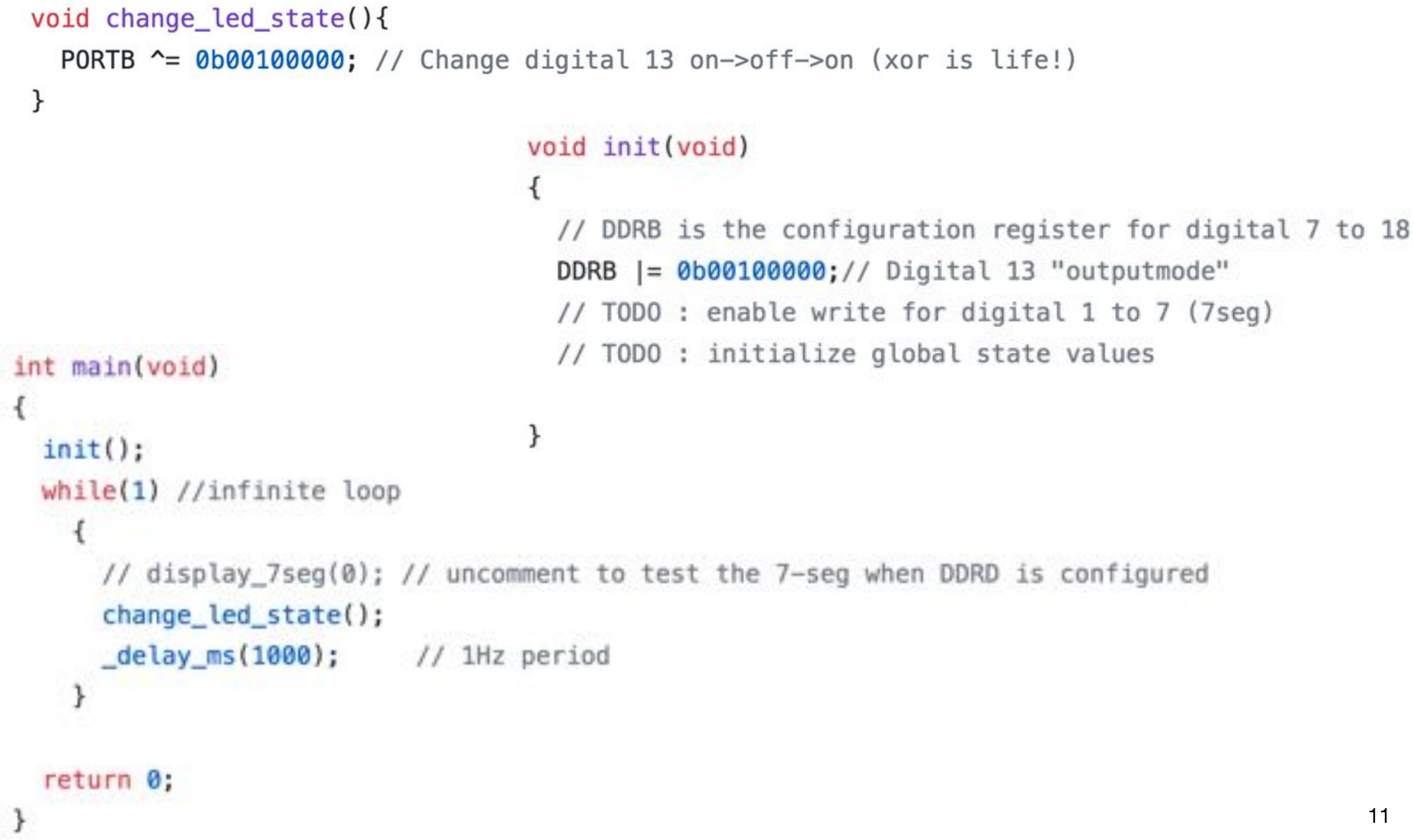
A journey into abstraction levels



Step-back

+ questions

Where do we start ?

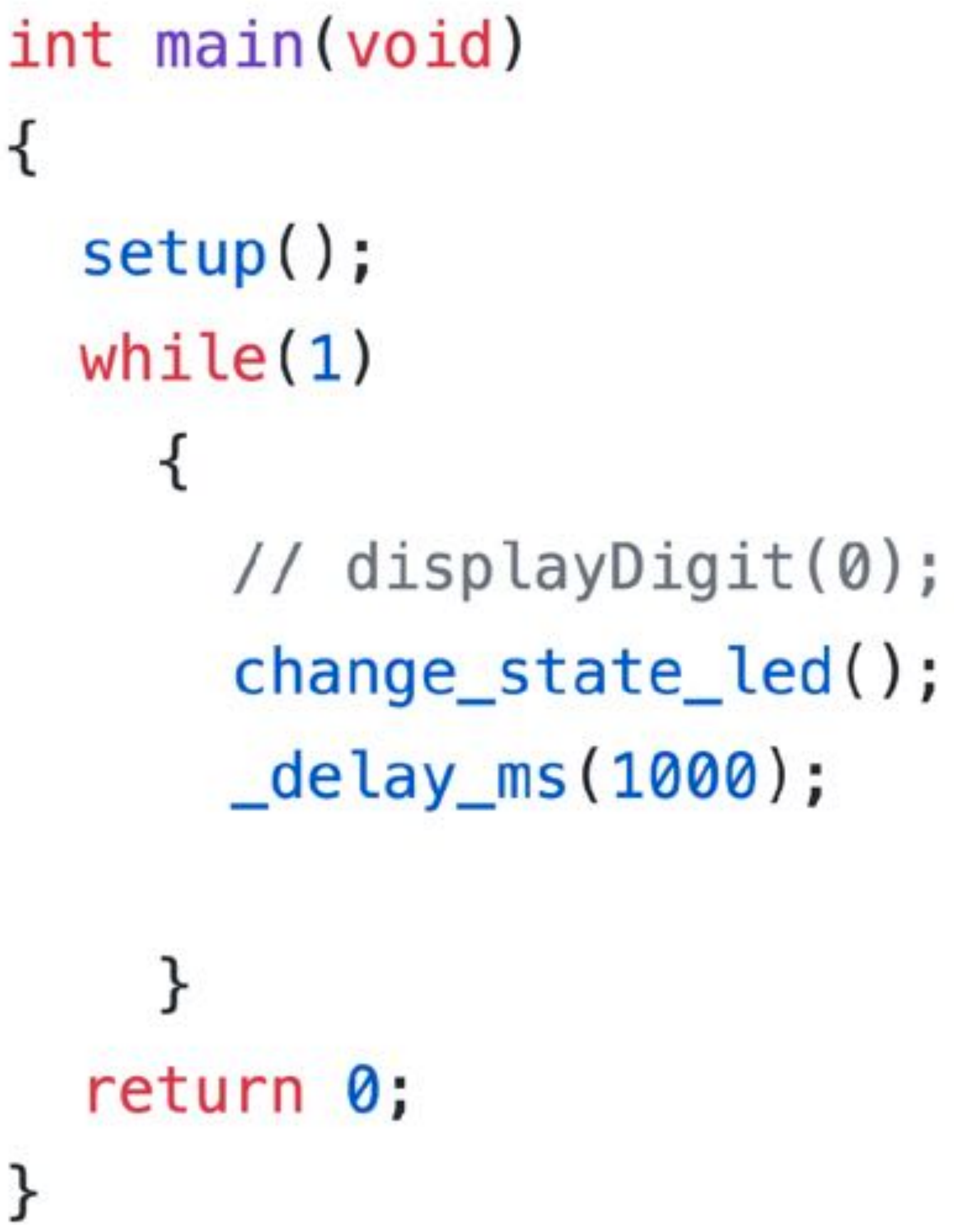


```
void change_led_state(){
  PORTB ^= 0b00100000; // Change digital 13 on->off->on (xor is life!)
}
```

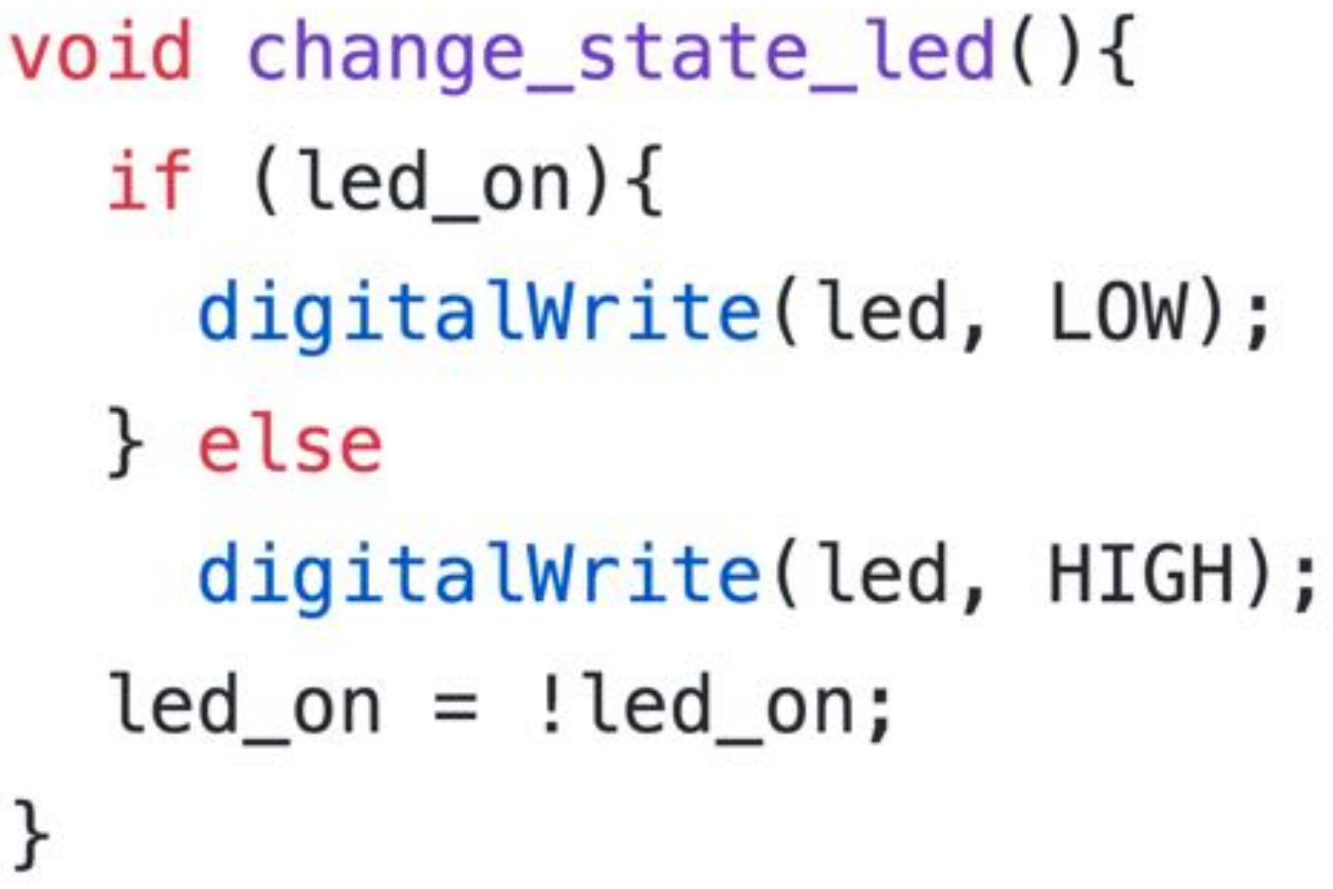
```
void init(void)
{
  // DDRB is the configuration register for digital 7 to 18
  DDRB |= 0b00100000;// Digital 13 "outputmode"
  // TODO : enable write for digital 1 to 7 (7seg)
  // TODO : initialize global state values

}
```

```
int main(void)
{
  init();
  while(1) //infinite loop
    {
      // display_7seg(0); // uncomment to test the 7-seg when DDRD is configured
      change_led_state();
      _delay_ms(1000);     // 1Hz period
    }

  return 0;
}
```

Library-based code

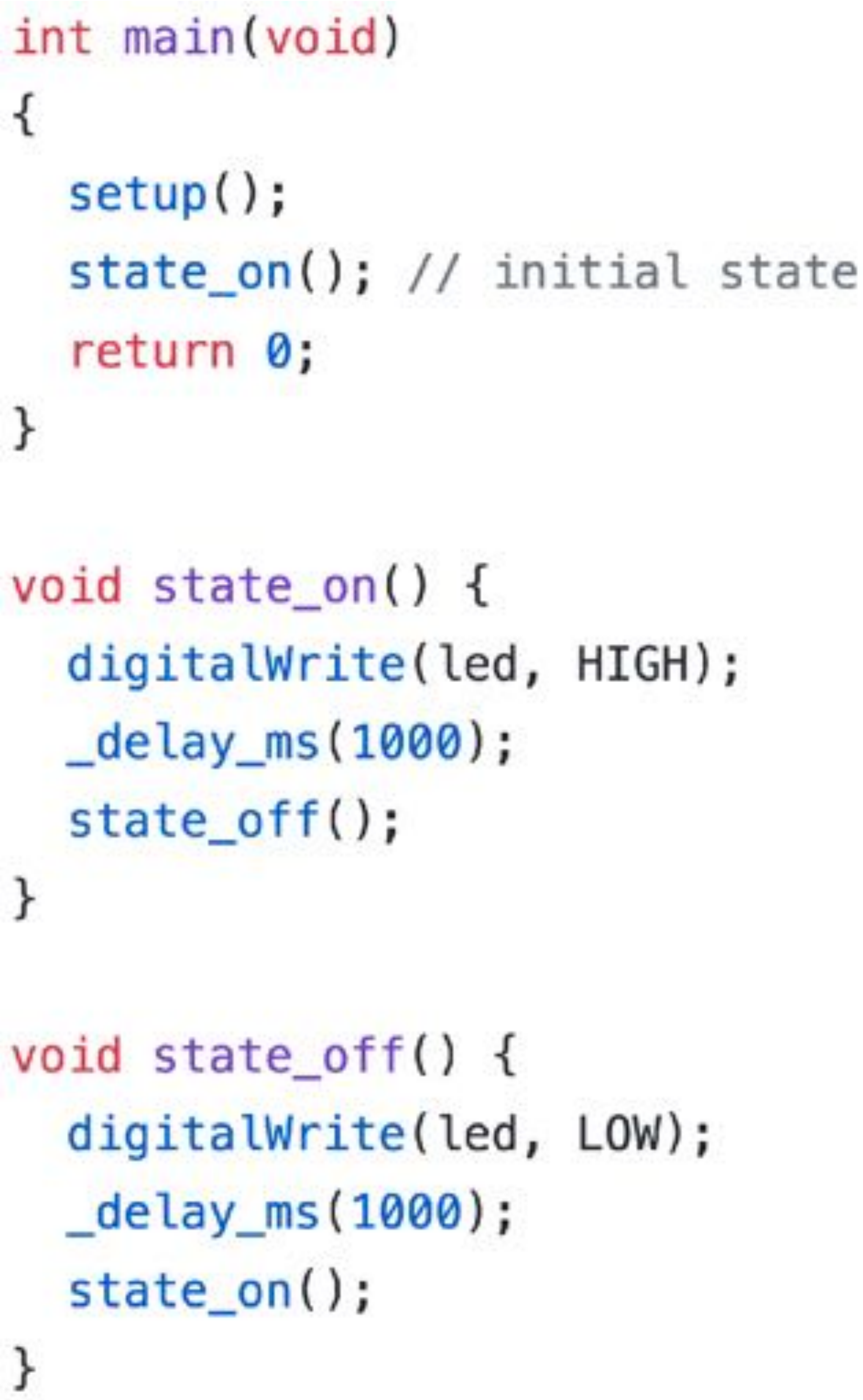


```
int main(void)
{
  setup();
  while(1)
    {
      // displayDigit(0);
      change_state_led();
      _delay_ms(1000);

    }
  return 0;
}
```

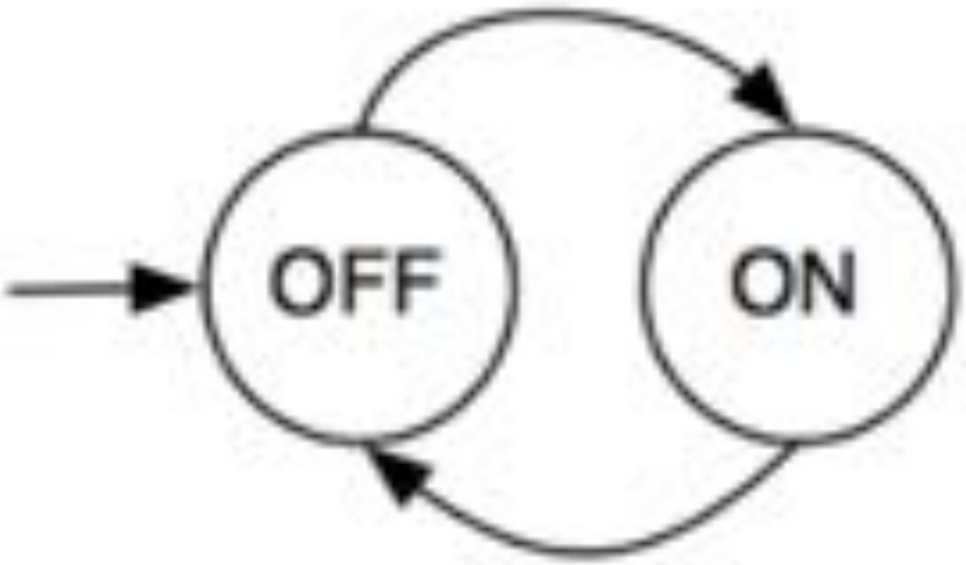


```
void change_state_led(){
  if (led_on){
    digitalWrite(led, LOW);
  } else
    digitalWrite(led, HIGH);
  led_on = !led_on;
}
```

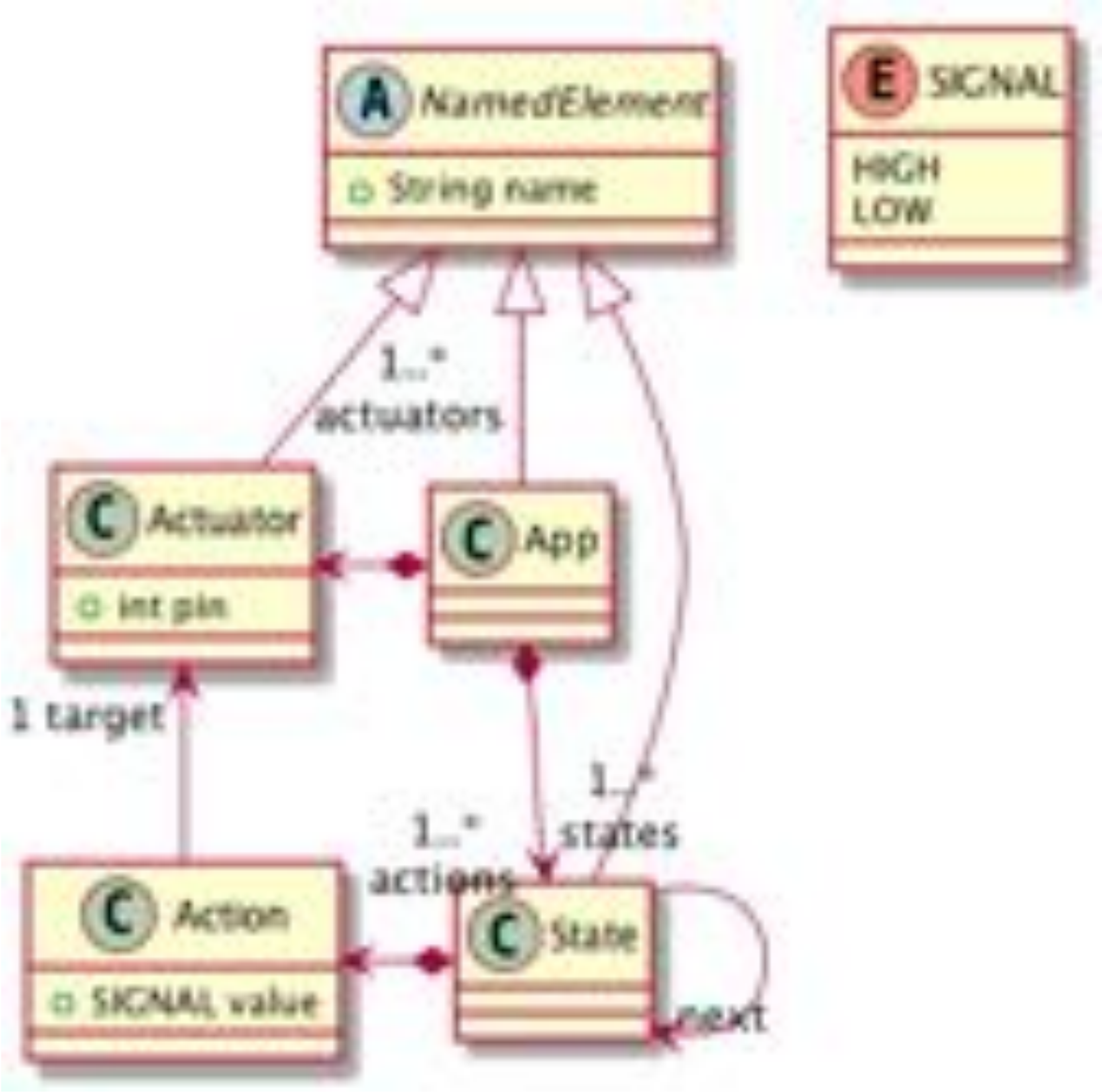
Programming a model



```
int main(void)
{
  setup();
  state_on(); // initial state
  return 0;
}

void state_on() {
  digitalWrite(led, HIGH);
  _delay_ms(1000);
  state_off();
}

void state_off() {
  digitalWrite(led, LOW);
  _delay_ms(1000);
  state_on();
}
```



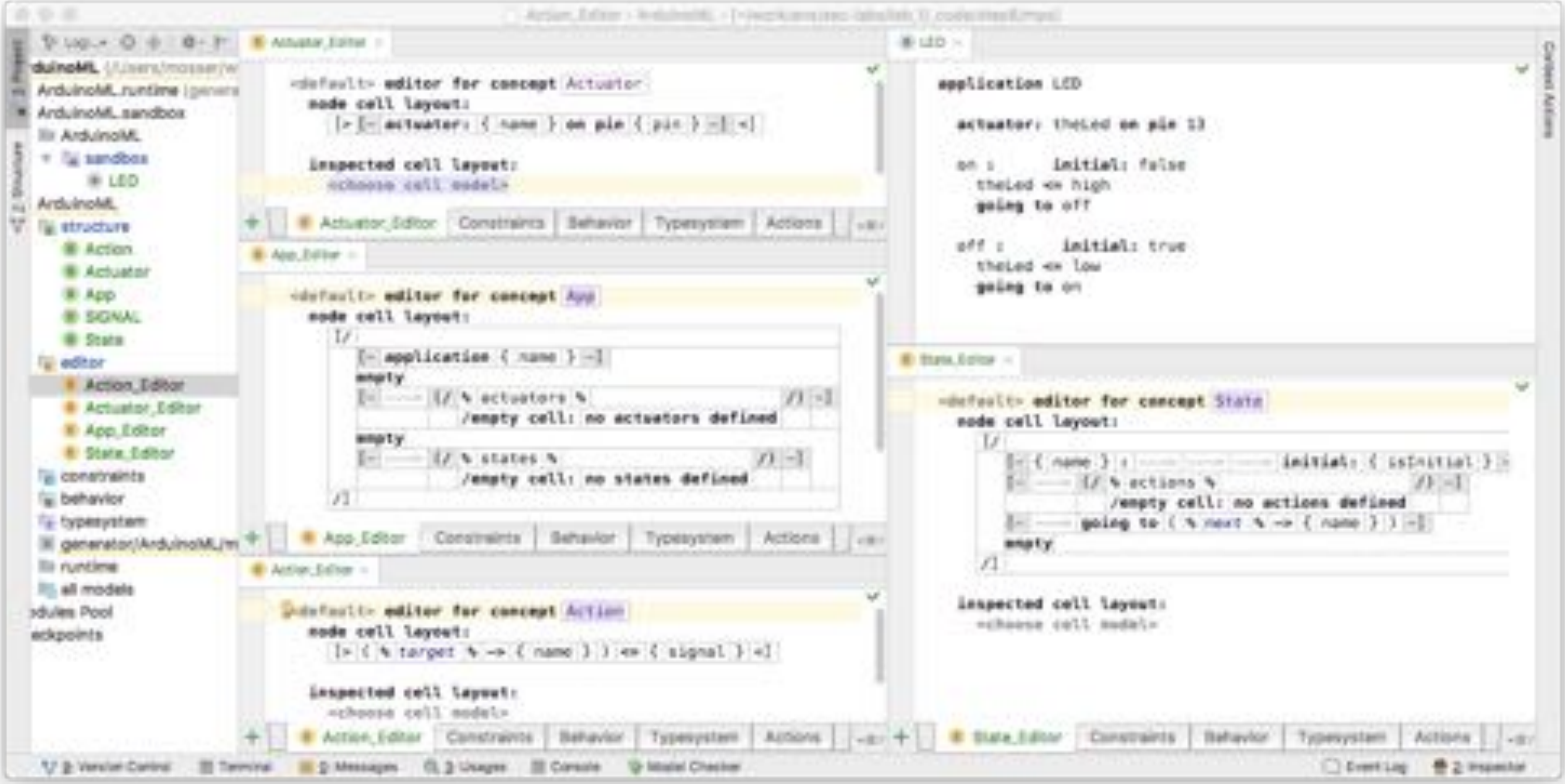
Modelling a Program



Tooling the model: Composing Apps

- Challenge: Express the LED example and the Counter one separately
    - **Compute the expected one instead of programming it**
- **How to create such a composition ?**
    - All the FSMs at the very same place (runtime composition semantic)
    - Creating an app containing all the FSMs separately
    - Merging the FSMs to assess global properties
    - ...

Designing a DSL



```
<default> editor for concept Actuator
  node cell layout:
    [> [- actuator: { name } on pin { pin } -] <]

  inspected cell layout:
    <choose cell model>
```

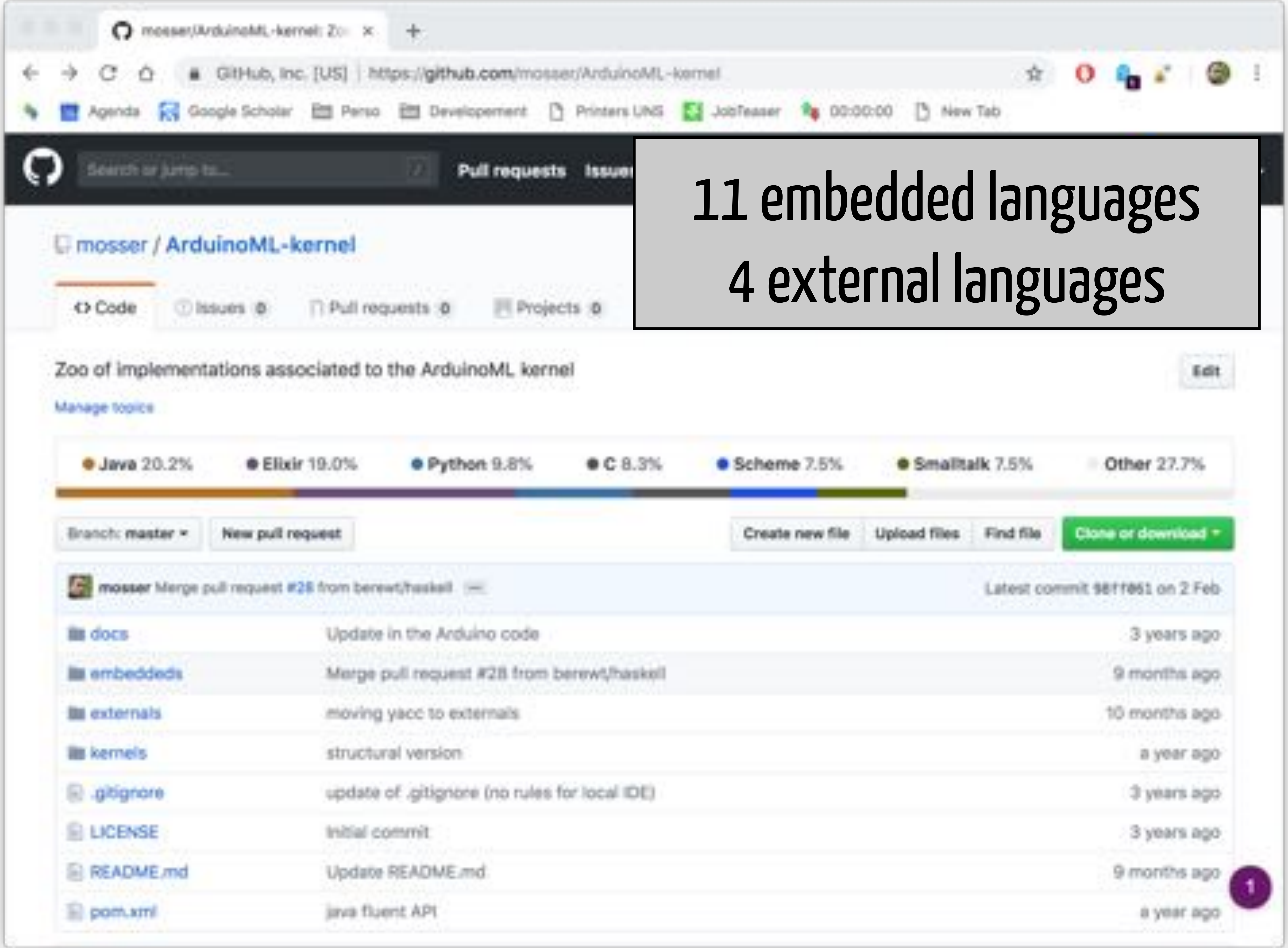
```
<default> editor for concept App
  node cell layout:
    [/
      [- application { name } -]
      empty
      [- ---- [/ % actuators %                          /] -]
                 /empty cell: no actuators defined
      empty
      [- ---- [/ % states %                     /] -]
                 /empty cell: no states defined
    /]
```

```
<default> editor for concept Action
  node cell layout:
    [> ( % target % -> { name } ) <= { signal } <]

  inspected cell layout:
    <choose cell model>
```

```
application LED

  actuator: theLed on pin 13

  on :        initial: false
    theLed <= high
    going to off

  off :       initial: true
    theLed <= low
    going to on
```

```
<default> editor for concept State
  node cell layout:
    [/
      [- { name } : ---- ---- ---- initial: { isInitial } -
      [- ---- [/ % actions %                    /] -]
                 /empty cell: no actions defined
      [- ---- going to ( % next % -> { name } ) -]
      empty
    /]

  inspected cell layout:
    <choose cell model>
```

11 embedded languages
4 external languages



mosser / ArduinoML-kernel

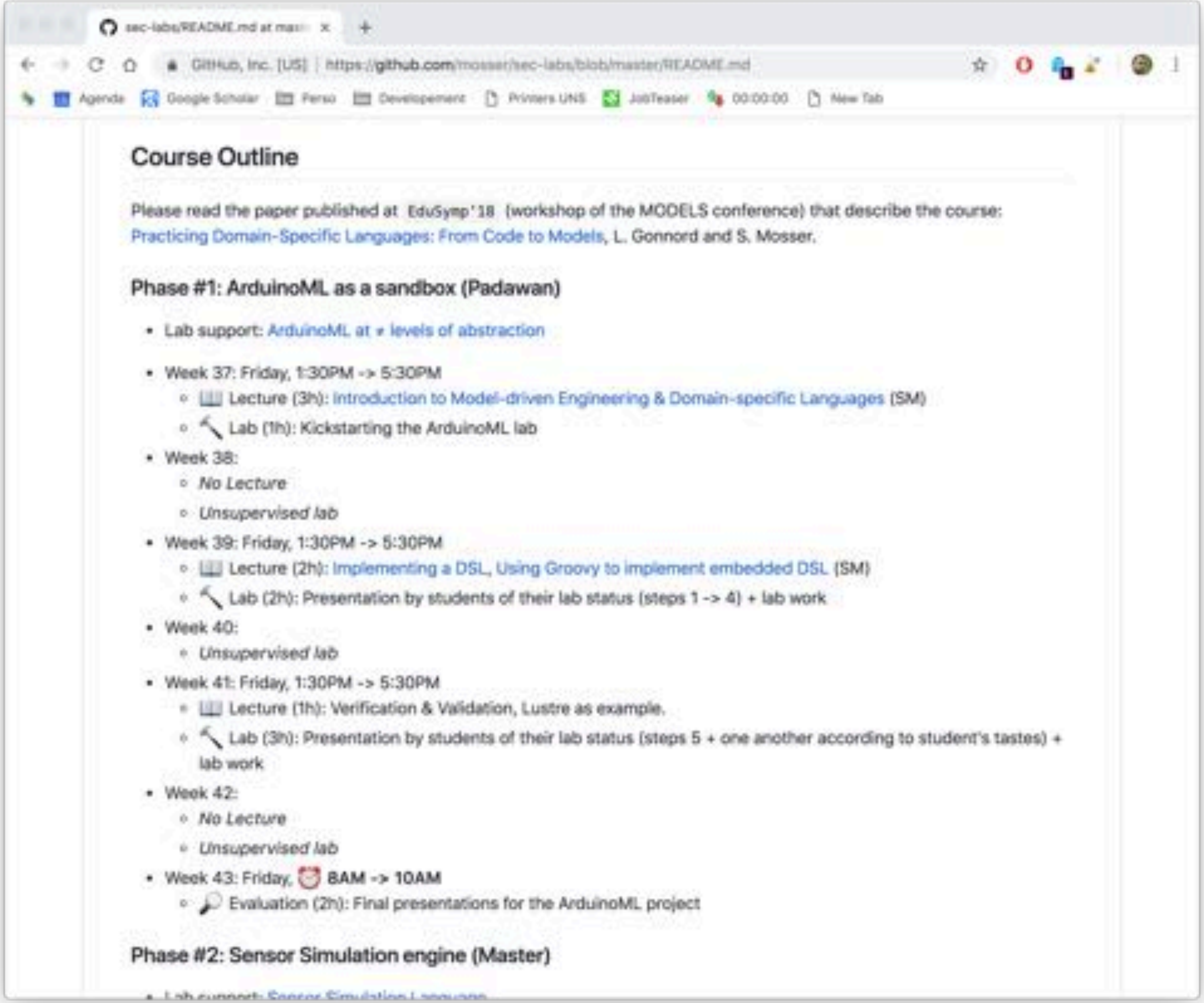
Zoo of implementations associated to the ArduinoML kernel

- Java 20.2%
- Elixir 19.0%
- Python 9.8%
- C 8.3%
- Scheme 7.5%
- Smalltalk 7.5%
- Other 27.7%

mosser Merge pull request #28 from berewt/haskell … Latest commit 98ff861 on 2 Feb

docs	Update in the Arduino code	3 years ago
embeddeds	Merge pull request #28 from berewt/haskell	9 months ago
externals	moving yacc to externals	10 months ago
kernels	structural version	a year ago
.gitignore	update of .gitignore (no rules for local IDE)	3 years ago
LICENSE	Initial commit	3 years ago
README.md	Update README.md	9 months ago
pom.xml	java fluent API	a year ago

https://github.com/mosser/ArduinoML-kernel


sec-labs/README.md at mas
GitHub, Inc. [US] | https://github.com/mosser/sec-labs/blob/master/README.md
Agenda
Google Scholar
Perso
Developement
Printers UNS
JobTeaser
00:00:00
New Tab

Course Outline

Please read the paper published at `EduSymp'18` (workshop of the MODELS conference) that describe the course: Practicing Domain-Specific Languages: From Code to Models, L. Gonnord and S. Mosser.

Phase #1: ArduinoML as a sandbox (Padawan)

- Lab support: ArduinoML at ∞ levels of abstraction
- Week 37: Friday, 1:30PM -> 5:30PM
  - 📖 Lecture (3h): Introduction to Model-driven Engineering & Domain-specific Languages (SM)
  - 🔨 Lab (1h): Kickstarting the ArduinoML lab
- Week 38:
  - *No Lecture*
  - *Unsupervised lab*
- Week 39: Friday, 1:30PM -> 5:30PM
  - 📖 Lecture (2h): Implementing a DSL, Using Groovy to implement embedded DSL (SM)
  - 🔨 Lab (2h): Presentation by students of their lab status (steps 1 -> 4) + lab work
- Week 40:
  - *Unsupervised lab*
- Week 41: Friday, 1:30PM -> 5:30PM
  - 📖 Lecture (1h): Verification & Validation, Lustre as example.
  - 🔨 Lab (3h): Presentation by students of their lab status (steps 5 + one another according to student's tastes) + lab work
- Week 42:
  - *No Lecture*
  - *Unsupervised lab*
- Week 43: Friday, ⏰ **8AM -> 10AM**
  - 🔎 Evaluation (2h): Final presentations for the ArduinoML project

Phase #2: Sensor Simulation engine (Master)

- Lab support: Sensor Simulation Language

https://github.com/mosser/sec-labs/

Implementation (grad)

ENS DE LYON

~ 10 students

Research-oriented institution

Practical labs + bibliographic study

Université Nice Sophia Antipolis | POLYTECH NICE-SOPHIA

> 40 students

Software Engineering

8 weeks project + exam

Key takeaways: No Pain, No Gain

Starting at a low level is "painful"

Pain helps to accept the modelling overhead

Students defend the MDE approach at the end!

Practicing Domain-Specific Languages: From Code to Models

Laure Gonnord
Univ Lyon, Université Claude Bernard Lyon 1
LIP, CNRS, ENS de Lyon, Inria
F-69342, LYON Cedex 07, France
Laure.Gonnord@ens-lyon.fr

Sébastien Mosser
Université Côte d'Azur, CNRS, I3S, France
Sophia Antipolis
mosser@i3s.unice.fr