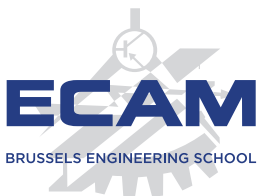
Automated Generation of Computer Graded Unit Testing-Based Programming Assessments for Education

Sébastien Combéfis[1,2] Guillaume de Moffarts[2]

[1]ECAM Brussels Engineering School (ECAM)

[2]Computer Science and IT in Education ASBL (CSITEd)

November 23, 2019



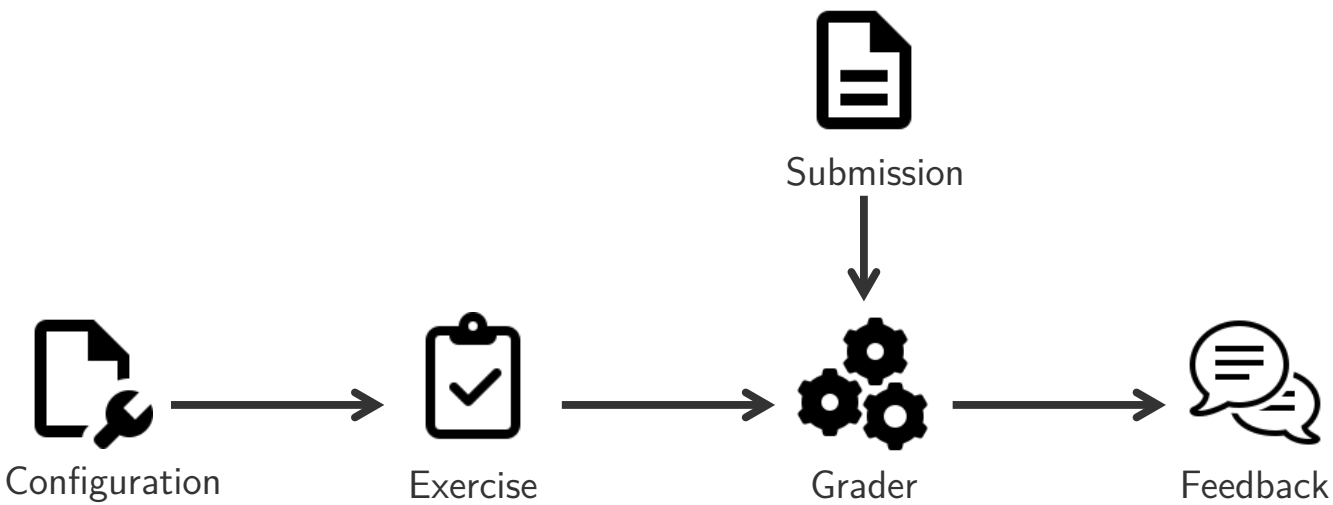
[CSEIT 2019, Zurich, Switzerland]

Context

- Automatic assessment of codes produced by learners
  - Demanded feature for learning management systems (LMS)
  - Mandatory when large number of students, as with MOOCs
- Code assessment consists in several processing
  - Compiling the code, that is, checking its syntax
  - Testing the code for correctness, by running some test cases
  - Generating a feedback to the learner, that helps him/her

Automatic Assessment Tool

- Tool to automatically generate coding exercises that can be...
  - ...solved in several programming languages
  - ...automatically graded
  - ...described with a single configuration file

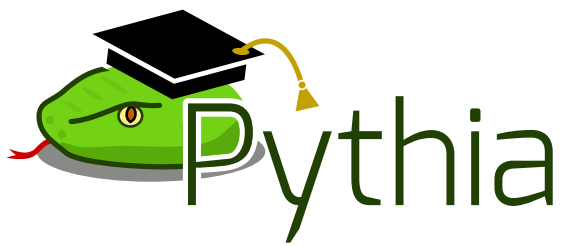


Motivation

- Automated code graders can be split in three categories
  - Code grader for programming competitions (online or offline)
  - Code evaluation for test-driven development
  - Code assessment for education
- Different goals for such automated code graders
  - Guarantees for execution environment and conditions
  - Correctness evaluation regarded test-cases execution
  - Feedback that supports learning

Pythia Platform

- Pythia platform automatically assesses student codes
  - Isolated secure sandboxes and constraints enforcement
  - Systematic way to test codes against tests suites
  - Tailored "intelligent" feedback generation

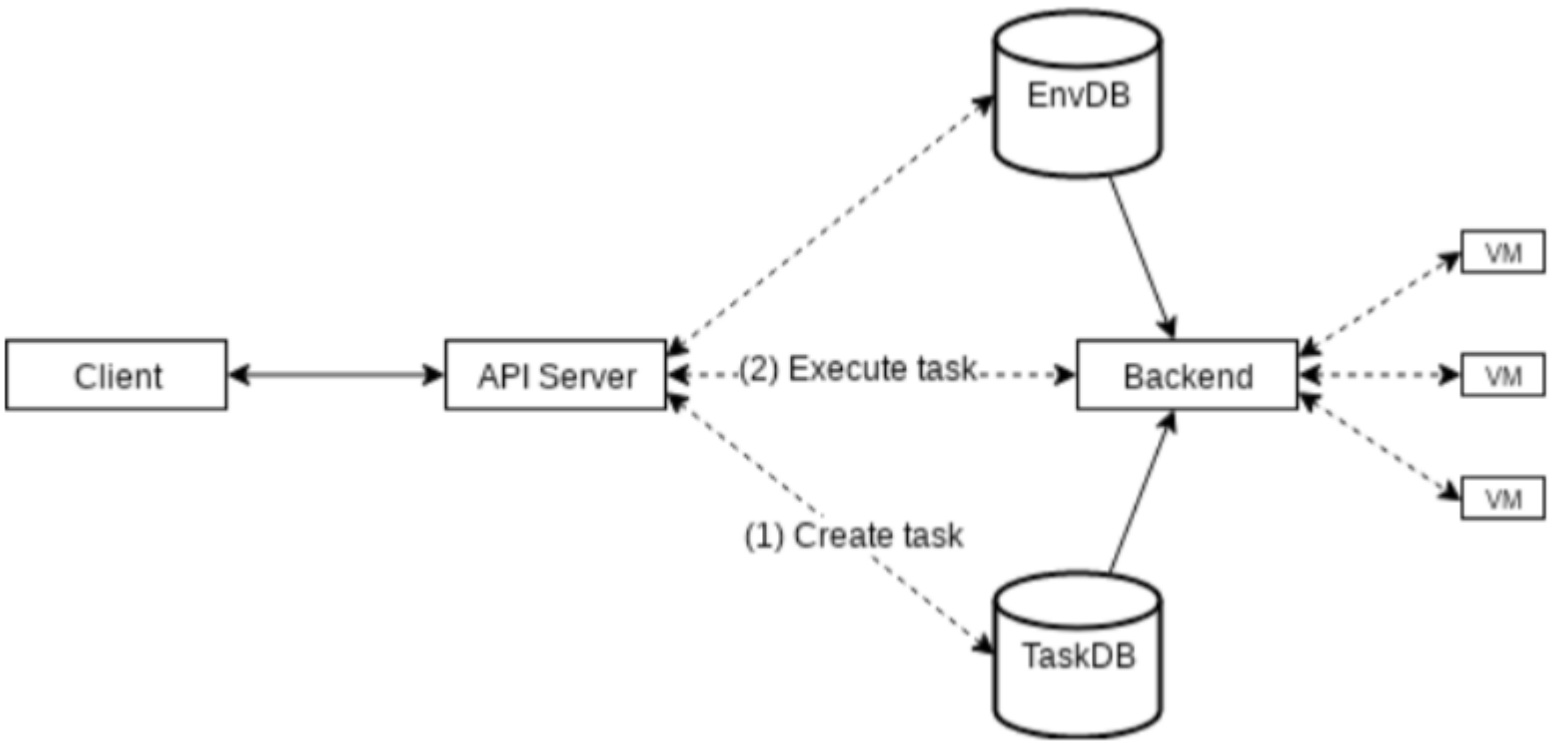


https://github.com/pythia-project

Architecture

- Distributed application with several components

  Developed with Go and uses UML virtual machines



Submission Grader

- Submission example for a unit testing-based exercise

  Fill the body of a function computing the sum of `a` and `b`

```
===INPUT EXAMPLE===
{
   "tid": "sub",
   "input": "{\"tid\": \"s001\", \"fields\": {\"f1\": \"return a\"}}"
}

==OUTPUT EXAMPLE===
{
   "tid": "s001",
   "status": "failed",
   "feedback": {
     "example": {
       "input": "(10,5)",
       "expected": "5",
       "actual": "10"
     },
     "message": "Have you subtracted the 2nd parameter?",
     "stats": {
       "succeeded": 2,
       "total": 14
     },
     "score": 0.14285715
   }
}
```

Task Execution (1)

- Pythia platform takes text input and produces text output

  Content and structure depends on the type of task

- A given kind of exercise always follow the same template

  Input-output and unit testing-based exercises do exist for now

- Unit-testing based exercises consists of four processes

  Highly configurable task templates with placeholders

Task Execution (2)

1. User input is preprocessed

   Student code template is filled

2. Tests suite generation

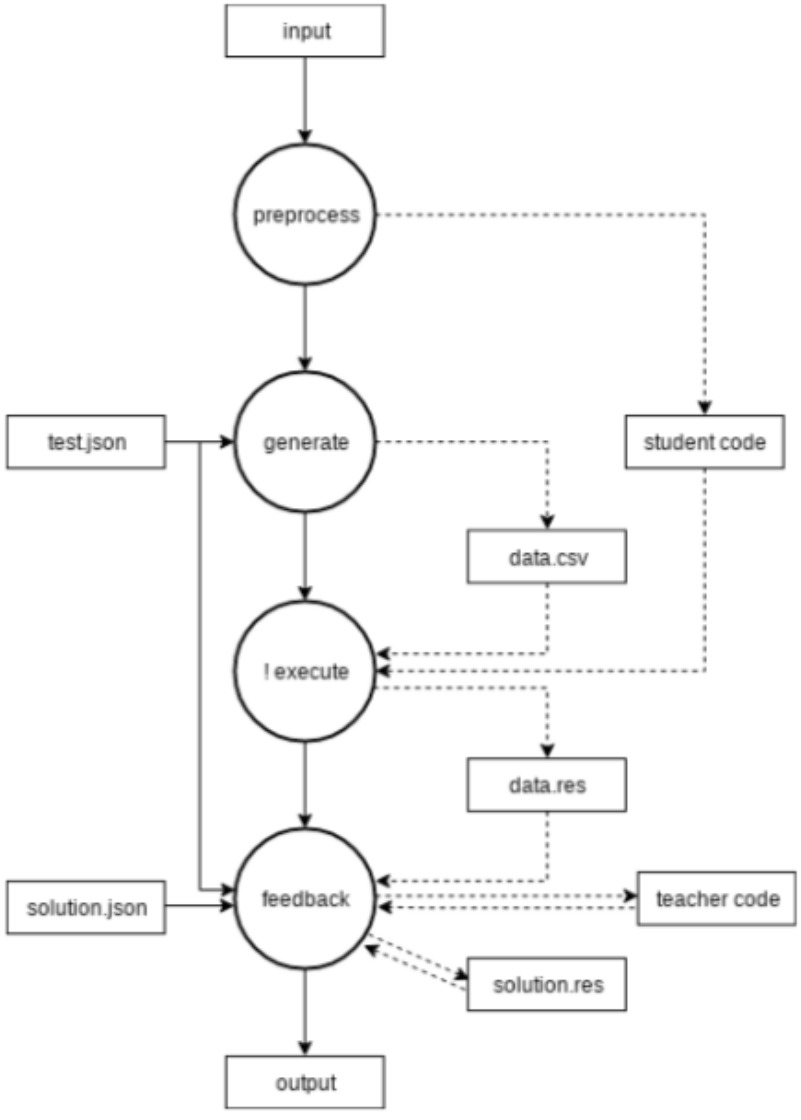
   Predefined and random tests

3. Student code is executed

   Unprivileged mode inside VM

4. Feedback is generated

   After having run correct code



Assessment Structure (1)

- Spec part of configuration defines exercise specification

  Precise signature of the function to complete

- Used to generate the code templates to execute student code

  Language-agnostic definition used to generate Python, Java, C...

```
{
  "spec": {
    "name": "sub",
    "args": [
      {
        "name": "a",
        "type": "int"
      },
      {
        "name": "b",
        "type": "int"
      }
    ],
    "return": "int"
  },
```

Assessment Structure (2)

- Test part with information for predefined and random tests

  Used to generate the tests suite used in execution phase

```
"test": {
  "predefined": [
    {
      "data": "(10, 5)",
      "feedback": {
        "10": "Have you subtracted the 2nd parameter?"
      }
    },
    {
      "data": "(7, 15)"
    },
    {
      "data": "(-1, 2)",
      "feedback": {
        "*": "Have you considered negative parameters?"
      }
    },
    {
      "data": "(12, 0)"
    }
  ],
  "random": {
    "n": 10,
    "args": [
      "int(-20,20)",
      "int(-20,20)"
    ]
  }
},
```

Assessment Structure (3)

- Solution contains one possible solution for the exercise

  Used to generate the correct answers for tests suite

- Executed after student code and before feedback generation

  Mandatory since random tests can be used

```
  "solution": {
    "f1": "return a - b"
  }
}
```

Exercise Generation

- Language-agnostic code for `preprocess`, `generate`, `feedback`

  Library written with the Go programming language

- Language-specific code for `execute` (student and teacher)

  One template per programming language

- Only code to be written by an instructor is a correct solution

  All the remainder elements come from the configuration

Conclusion and Further Work

- Pythia tool combines competition, TDD, education graders

  And support generation of "intelligent" feedbacks

- High flexibility thanks to a distributed architecture

  Possible to define new kinds of exercises

- Development of the API to propose more services

  JS snippet, LMS integration, code challenges for a company...

- Evaluation of the exercises against learning performances

  Measure whether generated feedback are useful and relevant