Simso web - Developper Documentation

Josué Alvarez

July 15, 2015

1 Preamble

This document is a non-exhaustive developer documentation, whose goal is to help future maintainers of the simso-web application to get started with the code. This document is written with the assumption that the developper is using an UNIX environment, has basic understanding of Javascript and Angular UI.

2 Getting Started

Introduction Simso web is a graphical interface built on top of simso. It runs as a full-client application (no server-side) written in javascript, and uses PypyJS (a javascript implementation of Pypy) to run Python in order to execute simso. The main frameworks/tools used for the front-end development are Angular JS and Bootstrap.

2.1 Getting the code

The code is available on github here : https://github.com/MaximeCheramy/simso-web. As simso-web embbeds its own version of simso, it is included in the form of a git submodule.

To setup your working copy, you have to run the following commands :

```
git clone https://github.com/MaximeCheramy/simso-web.git
cd simso-web/submodules/simso
git submodule init
git submodule update
```

2.2 Architecture

Overview The project is composed of 3 major components :

- The HTML / Javascript / AngularUI front-end.
- Simso : the component used to run the simulation.
- The python bridge between Simso and the Javascript application.

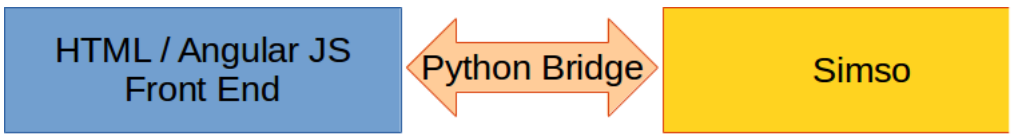

Figure 1: Simso web's major components

Use case In order to understand the structure of the code, we are going to introduce a typical simso-web use case :

1. The user sets up a configuration on the configuration view.
2. The user clicks on the "Run" button
   - A python configuration script is created using the parameters in the configuration view (file js/controlers/config-controler.py). This script is then executed and creates a python 'Configuration' object.
   - The 'run()' method of the Bridge is called (py/simso-bridge.py) and launches the Simso simulation.
   - Once this is done, a variable containing the results of the simulation is created in a way it can be read by the javascript application.
3. The user clicks the "Results" button : the results are now displayed. Some python functions (in py/simso-bridge.py) are used to aggregate results.

The HTML / Javascript / Angular UI front-end The front end is built upon the MVC architecture.

- The Model : the model files are contained within the js/services/ directory. They are called 'services'. The main services are :
  - conf-service.js : contains all the data related to the Configuration of the simulation. That data is going to be used to generate the configuration script.
  - pypy-service.js : contains all the code related to the initialisation of the virtual machine.
- The Views : the view files are contained in the partial/ folder. They are HTML files. There are 2 main views :
  - The Configuration view is the place where the user can setup the simulation parameters.
  - The Results view is the place where the user can see the simulation's results once it has been run.
- The Controlers : the controlers are located within the js/controlers/ directory.

The bridge The bridge is responsible for passing configuration to simso, gathering results and passing them back to javascript. The brige's source files are located in the py/ folder.

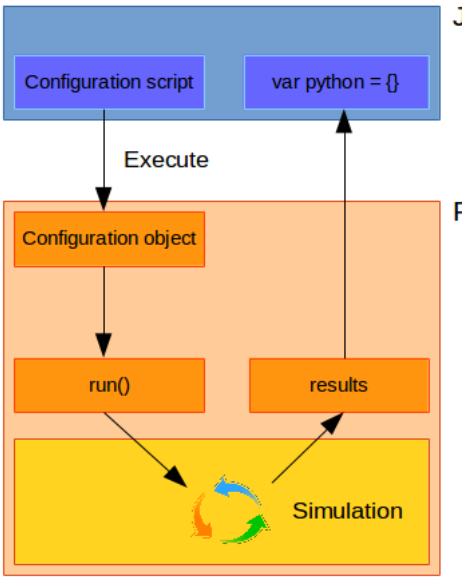

Figure 2: The python bridge

Simso Simso is integrated in simso-web as a submodule. Simso's documentation is available here : http://homepages.laas.fr/mcheramy/simso/doc.

3 Diving into the code

The goal of this section is to explain tricky or important parts of the code, as well as explaining some design choices.

3.1 The python virtual machine

As of this writting, Pypy.js doesn't have any form of documentation. In this section the functions used will get some documentation love.

Setup pass

Running a python script First make sure you have an access to pypyService. The code to execute python code should look like the following :

```
pypyService.vm.exec(code).then(function() {
   // Your code
}, logScriptErrors);
```

Where logScriptErrors is the function defined in config-controlers.js. This function will be called on error with an object as parameter containing a 'name', 'message' and 'trace'.

Managing imports As pypy.js will run in the user's browser with its own python environment, it is important to bundle all the needed modules. A listing of all the bundled modules can be found in lib/modules/index.json.

Passing variables to and from python We use the javascript global variable 'python' (defined in js/app.js) to pass variables from and to python. Passing variables from javascript to python is easy :

```
python["variable_name"] = myObject;
```

However, passing variables from python to javascript might be (a little bit) tricky in some cases.

```
import js
...
js.globals["python"]["variable_name"] = myObject;
```

The tricky part comes from the fact that myObject needs to be something pypy.js can convert to a javascript object. The easiest way to pass complex object is then to pass then as a hashmap (like you would do in javascript).

3.2 Simso integration in the configuration

Scheduler auto-detection mechanism Simso-web has a feature used to auto-detect all simso schedulers, the parameters they need for the themselves, the tasks and / or the processors. In Simso's code, every Scheduler wears the 'scheduler' decorator (declared in simso/schedulers/__init__.py). Here is an example :

```
@scheduler("simso.schedulers.CC_EDF",
           required_proc_fields = [
               { 'name': 'speed', 'type': 'float', 'default': '1.0' }
           ]
)
```

As executing all those decorators requires to import all the schedulers manually in pypy.js (which is not what we want), simso-web doesn't use them directly. Instead, it reads all the Schedulers descriptions in py/simso-schedulers.py. This file is generated automatically when executing the shell script tools/schedulers_discovery.sh.

When a scheduler is added/removed, the schedulers_discovery.sh script must be executed again !!

ETM detection All the Execution Time Models (ETM) must be declared in py/simso-etm.py. Here is an example of a ETM declaration :

```
{
```

```
        'name' : 'acet',
        'display_name' : 'ACET',
        'required_fields' : [],
        'required_proc_fields' : [],
        'required_task_fields' : [
          {
             'name' : 'acet',
             'display_name' : 'acet (ms)',
             'type' : 'float',
             'default' : '0'
          },
          {
             'name' : 'et_stddev',
             'display_name' : 'ET std dev (ms)',
             'type' : 'float',
             'default' : '0'
          }
        ]
},
```

3.3 Controlers hierarchy

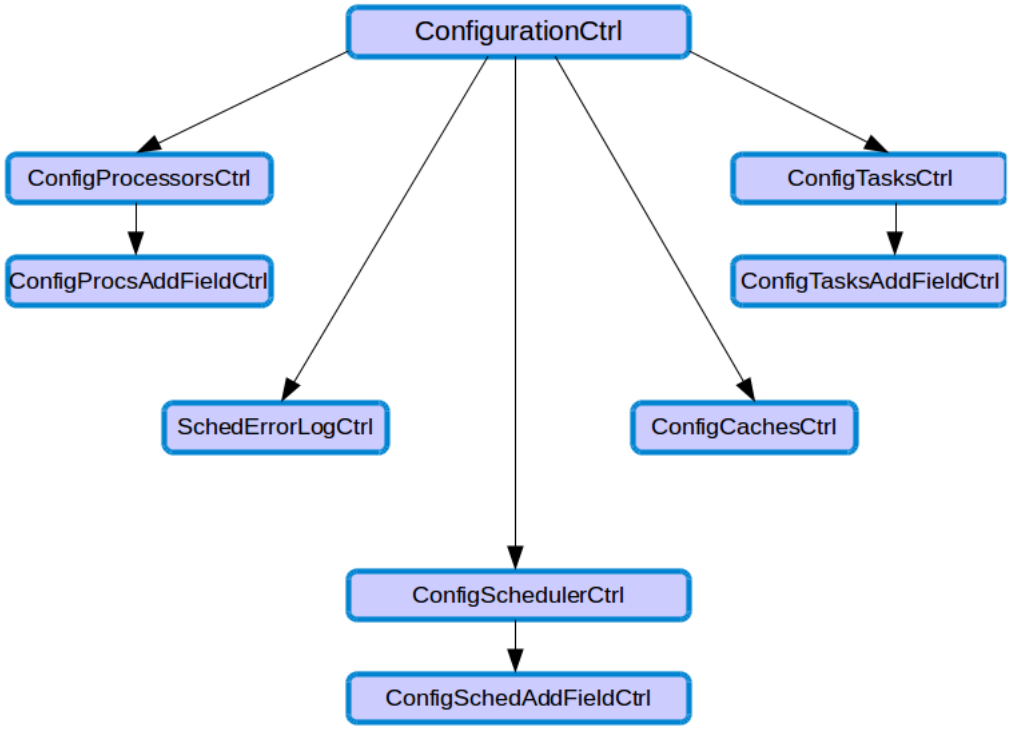

Figure 3: Configuration controlers scope hierarchy

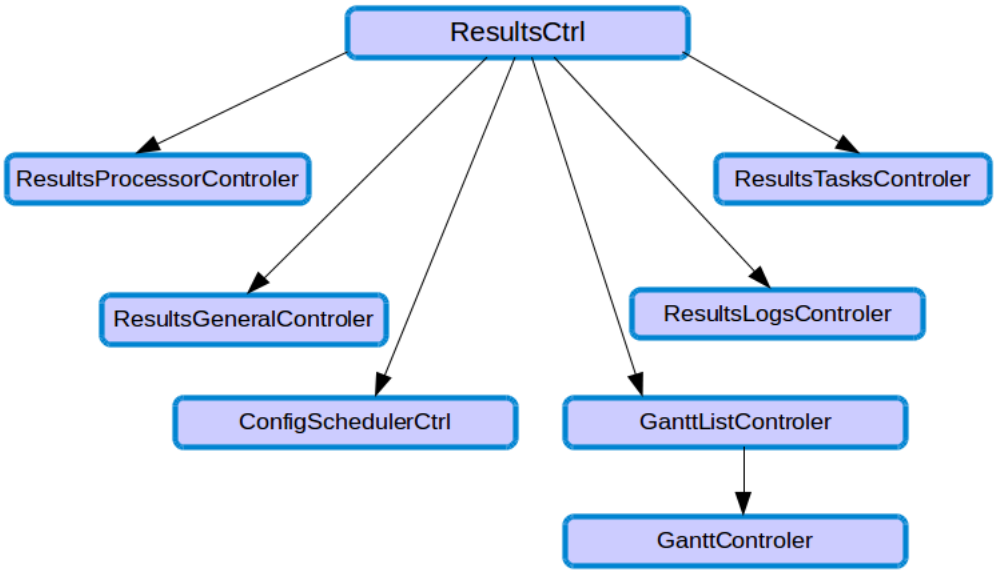

Figure 4: Results controlers scope hierarchy