INTO-CPS Maestro

Release 2.0.0 Alpha

Sep 24, 2020

Contents

For details on how to use Maestro, please see *User documentation*.

For details on how to assist in developing Maestro, please see the *Developer Documentation*.

For additional information on the INTO-CPS tool chain, please see https://github.com/INTO-CPS-Association/Documentation

CHAPTER 1

User documentation

Maestro is currently undergoing documentation updates. Currently, the user documentation consists of API documentation.

1.1 Getting Started

This page presents a getting started guide using the command-line interface of Maestro.

Additional information is available at *API*.

1.1.1 0. Environment

Maestro is built with Java 11, but it is expected that Java 8 will work as well.

1.1.2 1. Downloads

Download the latest coe jar from releases: https://github.com/INTO-CPS-Association/maestro/releases/latest

We will be running a co-simulation of the MassSpringDamper case study, described in https://github.com/INTO-CPS-Association/example-mass_spring_damper.

Download the two Mass Spring Damper FMUs exported from 20-sim: https://github.com/INTO-CPS-Association/example-mass_spring_damper/tree/master/FMUs/20-Sim

Place both jar and FMUs in the same folder.

1.1.3 2. Describe FMU Connections

Create the following `scenario.json` file in the same folder as the jar file:

```
{
    "fmus":{
        "{msd1}":"MassSpringDamper1.fmu",
        "{msd2}":"MassSpringDamper2.fmu"
    },
    "connections":{
        "{msd1}.msd1i.x1":[
            "{msd2}.msd2i.x1"
        ],
        "{msd1}.msd1i.v1":[
            "{msd2}.msd2i.v1"
        ],
        "{msd2}.msd2i.fk":[
            "{msd1}.msd1i.fk"
        ]
    },
    "logVariables":{
        "{msd2}.msd2i":[
            "x2",
            "v2"
        ]
    },
    "parameters":{
        "{msd2}.msd2i.c2":1.0,
        "{msd2}.msd2i.cc":1.0,
        "{msd2}.msd2i.d2":1.0,
        "{msd2}.msd2i.dc":1.0,
        "{msd2}.msd2i.m2":1.0
    },
    "algorithm":{
        "type":"fixed-step",
        "size":0.001
    },
    "loggingOn":false,
    "overrideLogLevel":"INFO"
}
```

1.1.4 4a. Running a Co-simulation using CLI

Open a terminal in the same folder and execute `java -jar coe-1.0.10-jar-with-dependencies.jar --configuration scenario.json --oneshot --starttime 0.0 --endtime 10.0`

Afterwards an *outputs.csv* file is available with the co-simulation results.

1.1.5 4b. Running a Co-simulation with Master Web Interface for Co-Simulation

This requires a bit more than execting a co-simulation using the CLI.

For this reason, a Python scrpt will be used as reference and explained in bits. `Full Python Script`

4b.1 Launch the COE

Launch the COE with a single argument, which makes it start up as a web server on port 8082: `java -jar coe-1.0.10-jar-with-dependencies.jar -p 8082`.

```
conn = http.client.HTTPConnection('localhost:' + str(port))
```

4b.2 Create a Session

It is necessary to create a session before conducting a co-simulation.

Example response: `{'sessionId': '5f439916-23f8-4609-9ff8-5f81408b9046'}`.

Python code:

```
print("Create session")
conn.request('GET', '/createSession')
response = conn.getresponse()
if not response.status == 200:
    print("Could not create session")
    sys.exit()

status = json.loads(response.read().decode())
print ("Session '%s', data=%s'" % (status["sessionId"], status))
```

4b.3 Initialize the co-simulation

Send the *scenario.json* to the server.

Example response: `[{"status":"Initialized","sessionId":"5f439916-23f8-4609-9ff8-5f81408b9046", "lastExecTime":0,"avaliableLogLevels":{"{msd2}.msd2i":[],"{msd1}.msd1i":[]}}]`.

Python code:

```
response = post(conn, '/initialize/' + status["sessionId"], "scenario.json")
if not response.status == 200:
    print("Could not initialize")
    sys.exit()

print ("Initialize response code '%d, data=%s'" % (response.status, response.read().
↪decode()))
```

4b.4 Optional: Connect Web Socket

At this stage, one can connect a web socket if so desired. This is currently not part of the scenario. See the *API* for more information.

4b.5 Run the Co-Simulation

The information passed as CLI arguments are now part of a *simulate.json* file:

```
{
  "startTime": 0,
  "endTime": 10
}
```

Send the *simulate.json* file to the server.

Example response: `{"status":"Finished","sessionId":"5f439916-23f8-4609-9ff8-5f81408b9046", "lastExecTime":1752}'`.

Python code:

```
def post(c, location, data_path):
    headers = {'Content-type': 'application/json'}
    foo = json.load(open(data_path))
    json_data = json.dumps(foo)
    c.request('POST', location, json_data, headers)
    res = c.getresponse()
    return res

response = post(conn, '/simulate/' + status["sessionId"], "simulate.json")
if not response.status == 200:
    print("Could not simulate")
    sys.exit()

print ("Simulate response code '%d, data=%s'" % (response.status, response.read().
↪decode()))
```

4b.6 Get the results

Retrieve the csv results and store in *result.csv*

Example response: CSV content (too large to show).

Python code:

```
conn.request('GET', '/result/' + status["sessionId"] + "/plain")
response = conn.getresponse()
if not response.status == 200:
    print("Could not receive results")
    sys.exit()

result_csv_path = "result.csv"
csv = response.read().decode()
print ("Result response code '%d" % (response.status))
f = open(result_csv_path, "w")
f.write(csv)
f.close()
```

4b.7 Destroy the session

Destroy the session and allow Maestro to clean up session data

Example reponse: 'Session 5f439916-23f8-4609-9ff8-5f81408b9046 destroyed'

Python code:

```
conn.request('GET', '/destroy/' + status['sessionId'])
response = conn.getresponse()
if not response.status == 200:
    print("Could not destroy session")
    sys.exit()
```

(continues on next page)

(continued from previous page)

```
print ("Destroy response code '%d, data='%s'" % (response.status, response.read().
↪decode()))
```

1.2 API

Maestro has multiple interfaces:

- Command Line Interface (CLI) *Command Line Interface*.
- Master Web Interface for Co-Simulation described in *Master Web Interface for Co-Simulation*.
- Slave Web Interface for Co-Simulation *Slave Web Interface for Co-Simulation*.

1.2.1 Command Line Interface

The command line interface is avaialble via the maestro jar file:

```
usage: coe
-c,--configuration <path>   Path to configuration file
-e,--endtime <time>         The start time of the simulation
-h,--help                   Show this description
-l,--load <path>            Attempt to load a single FMU
-o,--oneshot                Run a single simulation and shutdown
-p,--port <port>            The port where the REST interface will be
                            served
-r,--result <path>          Path where the csv data should be writting to
-s,--starttime <time>       The start time of the simulation
-v                          Verbose
-version,--version          Version
-x,--extract <type>         Extract values: 'script'
```

1.2.2 Master Web Interface for Co-Simulation

The master web interface is described in the following document: `:protocol`

It is also suggested to use the INTO-CPS Application to generate the required configuration and then examine the JSON to see the structure based on a given example.

1.2.3 Slave Web Interface for Co-Simulation

Efforts are being carried out to add the documentation of the slave web interface. Please ask if required.

CHAPTER 2

Developer Documentation

Maestro2 is currently under development in the 2.0.0-alpha branch.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search