Building Control Emulator

Release 0.0.1

Sep 29, 2021

Contents:

CHAPTER 1

Building Control Emulator platform

The building emulator is given as a Functional Mock-up Unit (FMU) and simulated using JModelica. JModelica, as the tool to simulate and analyze the building emulator behavior, has been packaged on a Ubuntu 16.04.5 LTS machine in a Docker container. Hence, in order to download, access and run the JModelica-specialized container, Docker needs to be installed on the host machine.

1.1 Docker Container

For Windows 10 and Mac OS, there are specific versions of Docker desktop, that is Docker desktop for Windows, and Docker desktop for Mac. On Ubuntu (Linux), installing Docker is less straight forward, and the procedure coudl follow the details below.

File Script to install Docker CE on Ubuntu, which presents what the docker installation site shows at Docker installation, can be used as helper to download and install Docker CE on Ubuntu.

```
#!/bin/bash

# Environment variables you need to set so you don't have to edit the script below.
DOCKER_CHANNEL=stable
DOCKER_COMPOSE_VERSION=1.18.0

# Update the apt package index.
sudo apt-get update

# Install packages to allow apt to use a repository over HTTPS.
sudo apt-get install -y \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common \
    vim

# Add Docker's official GPG key.
```

(continues on next page)

(continued from previous page)

```
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg |␣
↪sudo apt-key add -

# Verify the fingerprint.
sudo apt-key fingerprint 0EBFCD88

# Pick the release channel.
sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID
↪") \
  $(lsb_release -cs) \
  ${DOCKER_CHANNEL}"

# Update the apt package index.
sudo apt-get update

# Install the latest version of Docker CE.
sudo apt-get install -y docker-ce

# Allow your user to access the Docker CLI without needing root.
sudo /usr/sbin/usermod -aG docker $USER

# Install Docker Compose.
curl -L https://github.com/docker/compose/releases/download/${DOCKER_COMPOSE_VERSION}/
↪docker-compose-`uname -s`-`uname -m` -o /tmp/docker-compose
sudo mv /tmp/docker-compose /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo chown root:root /usr/local/bin/docker-compose
```

The script also installs Docker Composer, used to define and run a multi-container Docker application. See Compose overview.

Warning. To be able to run the Docker CLI without needing root, you need a reboot.

CHAPTER 2

JModelica Docker

2.1 Getting the JModelica emulator docker image

Note. The following procedures are related to Mac OS and Ubuntu.

Once Docker desktop is installed on the host computer, to get access to the JModelica container, one could follow the steps below. Details on the Docker commands can be found on the Docker documentation page.

1. Open a terminal window.
2. At the terminal prompt type

```
docker pull laurmarinovici/building_control_emulator:latest
```

The docker image will be downloaded on the host computer.

3. To inspect the Docker images downloaded type

```
docker images
```

should return a list of Docker images, which should include something similar to

4. To instantiate the Docker container, run

```
docker run -it --rm -p="127.0.0.1:5000:5000" \
         --mount type=bind,source=<path to host computer folder to bind with␣
↪container folder>,destination=<path to folder in the container bound to host folder>
↪ \
         --network=host --name=<container name> <image name> bash

Normally, the host computer folder bound to a folder within the container would be␣
↪the folder that contains the models and the running scripts (developed or␣
↪downloaded from the github repository).
```

5. Once the container has been created, it should show up listed when running

```
docker ps -a
```

2.2 Inside the JModelica Docker container

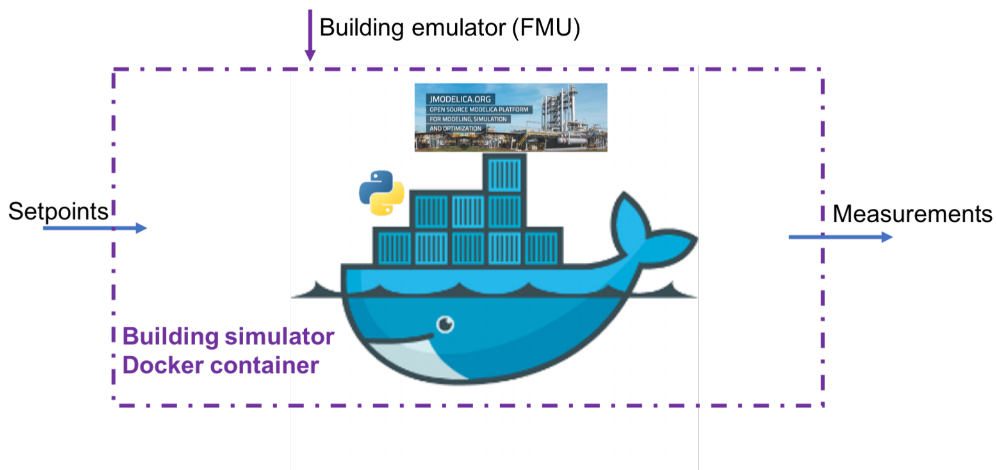

Fig. 1: Figure 1. Emulator Docker diagram

JModelica Docker container is build on an Ubuntu distribution version *16.04.6 LTS (Xenial Xerus)*. It contains **'JModelica'_** and the neccessary Python modules:

- PyModelica - for compiling Modelica models intu FMUs
- PyFMI - for loading and interacting with the FMU representing the building emulator

Inside the *JModelica Docker container*, the building emulator is loaded and simulated/controlled using a REST (REpresentational State Transfer) API.

Class *emulatorSetup* has been implemented to define the REST API requests to perform functions such as advancing the simulation, retrieving test case information, and calculating and reporting results.

Code documentation - *emulatorSetup.py*

- *Acquire the list of inputs the emulator accepts as control signals*

  The emulator inputs are pairs of 2 values for each control signal:

  – *<name>_activate* - that can take 0 or 1 values indicating that particular input is going to be used for control with the given value rather than the default value

  – *<name>_u* - that represents the actual input value that the control designer calculates

class emulatorSetup.**emulatorSetup**(*fmuPath*, *fmuStep*)
Class to setup the building emulator (FMU) simulation.

get_inputs()
Returns a list of control input names.
Parameters **None** –
Returns **inputs** – List of control input names.
Return type list

- *Acquire the list of measurements exposed by the emulator*

class emulatorSetup.**emulatorSetup**(*fmuPath*, *fmuStep*)
Class to setup the building emulator (FMU) simulation.

get_measurements()
Returns a list of measurement names.
Parameters **None** –
Returns **measurements** – List of measurement names.
Return type list

- *Advance the emulator simulation one step further after providing a set of control inputs to it with*

class emulatorSetup.**emulatorSetup**(*fmuPath*, *fmuStep*)
Class to setup the building emulator (FMU) simulation.

advance(*u*)
Advances the test case model simulation forward one step.
Parameters **u** (*dict*) – Defines the control input data to be used for the step. {<input_name> : <input_value>}
Returns **y** – Contains the measurement data at the end of the step. {<measurement_name> : <measurement_value>}
Return type dict

- *Obtain the name of the emulator*

class emulatorSetup.**emulatorSetup**(*fmuPath*, *fmuStep*)
Class to setup the building emulator (FMU) simulation.

get_name()
Returns the name of the FMU being simulated.
Parameters **None** –
Returns **name** – Name of FMU being simulated.
Return type str

- *Obtain the simulation time step in seconds*

class emulatorSetup.**emulatorSetup**(*fmuPath*, *fmuStep*)
Class to setup the building emulator (FMU) simulation.

get_step()
Returns the current simulation step in seconds.

- *Set the simulation time step in seconds*

class emulatorSetup.**emulatorSetup**(*fmuPath*, *fmuStep*)
Class to setup the building emulator (FMU) simulation.

set_step(*step*)
Sets the simulation step in seconds.
Parameters **step** (*int*) – Simulation step in seconds.
Returns
Return type None

- *Obtain full trajectories of measurements and control inputs*

class emulatorSetup.**emulatorSetup**(*fmuPath*, *fmuStep*)
Class to setup the building emulator (FMU) simulation.

get_results()
Returns measurement and control input trajectories.
Parameters **None** –

Returns Y – Dictionary of measurement and control input names and their trajectories as lists. {'y':{<measurement_name>:<measurement_trajectory>}, 'u':{<input_name>:<input_trajectory>}}

Return type dict

- *Obtain key performance indicator (kpi)*

class emulatorSetup.**emulatorSetup**(*fmuPath*, *fmuStep*)

Class to setup the building emulator (FMU) simulation.

get_kpis()

Returns KPI data.

Requires standard sensor signals.

Parameters

- **None** –
- **Returns** –
- **kpi** (*dict*) – Dictionary containing KPI names and values. {<kpi_name>:<kpi_value>}

Script *startREST* instantiate the building emulator by loading the desired FMU file and setting up the length of the time interval (in seconds) for which the emulator will run until finishing or being interrupted to receive an external control action. It also opens up the communication channels through which HTTP requests can be made to access the building emulator. The scripts should be called using:

```
python startREST.py -p ./models/wrapped.fmu -s 60
```

or

```
python startREST.py --fmuPath=./models/wrapped.fmu --fmuStep=60
```

Code documentation - *startREST.py*

class startREST.**Advance**(***kwargs*)

Interface to advance the test case simulation.

post()

POST request with input data to advance the simulation one step and receive current measurements.

class startREST.**Inputs**(***kwargs*)

Interface to test case inputs.

get()

GET request to receive list of available inputs.

class startREST.**Measurements**(***kwargs*)

Interface to test case measurements.

get()

GET request to receive list of available measurements.

class startREST.**Results**(***kwargs*)

Interface to test case result data.

get()

GET request to receive measurement data.

class startREST.**KPI**(***kwargs*)

Interface to test case KPIs.

get()
GET request to receive KPI data.

class startREST.**Name**(***kwargs*)
Interface to test case name.

get()
GET request to receive test case name.

CHAPTER 3

Building emulator examples

3.1 How to run a simple example

On Building Control Emulator Github repository at *https://github.com/SenHuang19/BuildingControlEmulator*:

- folder *emulatorExamples* contains:
  - *emulatorSetup.py* - to implement the emulatorSetup class
  - *startREST.py* - to load the building emulator/FMU and start the REST server
  - folder *models* that includes the building emulators given as FMU files

  This folder needs to be bound to a folder inside the container to have access to the FMU to simulate.

- folder *simulationExamples* contains:
  - *runSimulation.py* - script to be run from the host computer to simulate the emulator inside the docker, control it if need be, get results, or whatever else the developer wants to add. This script is to be called (as seen later in the methodology) using

```
python runSimulation.py -u "http://0.0.0.0:5000" -d 200 -o 0 -l 1200 -s 300
```

or

```
python runSimulation.py --url="http://0.0.0.0:5000" --dayOfYear=200 --dayOffset=0␣
↪--simDuration=1200 --fmuStep=300
```

where

  - *-u*, *–url* represents the URL of the Docker that runs the REST server has. In this case it is *http://0.0.0.0:5000* because the emulator docker runs locally;
  - *-d*, *–dayOfYear* represents the day of year when the emulator simulation starts;
  - *-o*, *–dayOffset* represents the offset in seconds from second zero of the day when the simulation starts in the day previously set;

- -*l*, –*simDuration* represents the entire simulation duration in seconds;
- -*s*, –*fmuStep* represents the period for which the FMU is being simulated before stopping and/or waiting for external control; this value would actually overwrite the *fmuStep* given when instantiating the *emulatorSetup* class.

3.2 Methodology

Disclaimer. This procedure has been tested and worked well on a Mac or Linux machine with Docker installed as presented in Docker Container.

1. On a computer with docker installed, open a terminal and pull the building cnotrol emulator image.

```
docker pull laurmarinovici/building_control_emulator:latest
```

2. Clone the repository in to your home directory: /home/*networkID*.

```
git clone https://github.com/SenHuang19/BuildingControlEmulator
```

3. To instantiate the Docker container, run

```
docker run -it --rm -p="127.0.0.1:5000:5000" \
        --mount type=bind,source=/home/*networkID*/BuildingControlEmulator/
↪emulatorExamples/,destination=/mnt/examples \
        --name=jmodelica_docker laurmarinovici/building_control_emulator:latest bash
```

where /home/*networkID*/ is the local folder where the building control emulator Github repository has been cloned to, and */mnt/examples* is just a folder on the already started *jmodelica_container*.

4. At the opened terminal inside the container:

```
cd /mnt/examples
```

5. Run

```
python startREST.py --fmuPath=./models/wrapped.fmu --fmuStep=60
```

The app should start showing

```
* Serving Flask app "startREST" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

6. At a different terminal

```
cd /Users/mari009/PNNL_Projects/GitHubRepositories/BuildingControlEmulator/
↪simulationExamples

python runSimulation.py --url="http://0.0.0.0:5000" --dayOfYear=200 --dayOffset=0 --
↪simDuration=1200 --fmuStep=300
```

7. After 4 300-second intervals, within which the building emulator is simulated, the simulation ends, and the user can observe the following output files:

- in <..>*/BuildingControlEmulator/simulationExamples*:
    - *results.csv* containing some sample measurements taken at the end of each 300-second interval
    - *measurementsList.csv* containing a list of all the measurements exposed for the building model
    - *controlInputsList.csv* containing a list of control signals that can be by an external control at the beginning of each 300-second interval to overwrite or not the default control signals that come with the building model:
        * *<control signal name>_activate* - flag that would signal to the emulator whether that control value should be overwrtten (when flag is set to 1) or disregarded (flag is set to 0)
        * *<control signal name>_u* - the actual value of the control signal for that particular time
- in <..>*/BuildingControlEmulator/emulatorExamples*:
    - *<FMU name>_result.mat - THIS STILL NEED TO BE WORKED OUT*

3.3 Building emulator measurements nomenclature

Measurement name	Description (floor # = {1, 2, 3}, zone # = {1, 2, 3, 4, 5})	Unit
time	time of measurement	second
TOutDryBul_y	actual outside/ambient temperature	Kelvin
PChi_y	chiller power consumption	Watt
PPum_y	pump power consumption	Watt
PBoiler_y	boiler gas consumption	Watt
floor#_Pfan_y	fan power consumption on floor #	Watt
floor#_conCoiEco_ oveTMix_Sig_y	actual AHU mixed air temperature on floor #	Kelvin
floor#_conCoiEco_ oveTRet_Sig_y	actual AHU return air temperature on floor #	Kelvin
floor#_conCoiEco_ oveTSup_Sig_y	actual AHU SUPPLY air temperature on floor #	Kelvin
floor#_conCoiEco_ mSup_y	actual AHU SUPPLY air flow rate on floor #	kg/s
floor#_conFan_FanSpeed _Sig_y	AHU speed on floor #	Fraction
floor#_conFan_OvePre _Sig_y	AHU static pressure on floor #	Pa
floor#_conFan_ OvePreSetPoi_Sig_y	AHU static pressure set point on floor #	Pa
floor#_zon#_TSupAir_y	actual discharge air temperature in zone # on floor #	Kelvin
floor#_zon#_mSupAir_y	actual air flow in zone # on floor #	Kg/s
floor#_zon#_ TSetRooCoo_u	cooling temperature set point in zone # on floor #	Kelvin
floor#_zon#_ TSetRooHea_u	heating temperature set point in zone # on floor #	Kelvin
floor#_zon#_OccSch	occupant schedule of zone # on floor #	Binary
floor#_zon#_PPD	ppd of zone # on floor #	%

3.4 Building emulator controllable signals nomenclature

Signal name	Description (floor # = {1, 2, 3}, zone # = {1, 2, 3, 4, 5})	Unit
floor#_onCoiEco_Eco_ovePos_u	set point for damper position at the AHU level on floor #	fraction
floor#_onCoiEco_oveBlockEco_ovePos_u	damper position at the AHU level on floor #	fraction
floor#_conCoiEco_oveTMix_oveSig_y	mixed air temperature sensor measurement at the AHU level on floor #	Kelvin
floor#_oveTout_oveSig_u	outside/ambient temperature sensor measurement at AHU level on floor #	Kelvin
floor#_conCoiEco_oveTRet_oveSig_y	return air temperature sensor measurement at the AHU level on floor #	Kelvin
floor#_conCoiEco_oveTSupSetPoi_oveSig_u	set point for supply air temperature at AHU level on floor #	Kelvin
floor#_conCoiEco_oveTSup_oveSig_y	supply air temperature sensor measurement at the AHU level on floor #	Kelvin
floor#_conCoiEco_oveBlockCooCoi_oveLeakage_u	cooling coil leakage at AHU level on floor #	Fraction
floor#_conCoiEco_oveBlockCooCoi_ovePos_u	cooling coil valve position at AHU level on floor #	Fraction
floor#_conCoiEco_CooCoi_oveSig_u	position set point for cooling coil valve at AHU level on floor #	Fraction
floor#_conFan_OvePre_oveSig_u	static pressure sensor measurement at AHU level on floor #	Pa
floor#_conFan_OvePreSetPoi_oveSig_u	static pressure set point at AHU level on floor #	Pa
floor#_hvac_oveBlockDamper_ovePos_u	air flow relative to max in zone # on floor #	fraction
floor#_hvac_oveBlockHeaCoi_ovePos_u	reheat valve position in zone # on floor #	fraction
floor#_zon#_oveTRooAir_u	room air temperature sensor measurement in zone # on floor #	Kelvin
floor#_zon#_oveTSetRooCoo_u	cooling temperature set point in zone # on floor #	Kelvin
floor#_zon#_oveTSetRooHea_u	heating temperature set point in zone # on floor #	Kelvin
floor#_zon#_oveOcc	occupant schedule in zone # on floor #	Binary
oveTChWSet	set point of the chilled water leaving the chilelr	Kelvin

3.5 List of examples

The following examples should be found in */emulatorExamples/models/*:

- *wrapped.fmu* - just for exemplifying sake
- *LargeOffice - NEED DESCRIPTION*
- *LargeOfficeFDD - NEED DESCRIPTION*

CHAPTER 4

Building emulator controlled using the adaptive MPC example

4.1 How to run a building adaptive MPC example

For those who have access to the adaptive MPC repository, here are the steps to run an integrated building emulator and adaptive MPC case.

1. Download the Docker images

- Building emulator Docker image at *laurmarinovici/building_control_emulator:latest*
- Julia 1.2.0 on Ubuntu 18.04 image at *laurmarinovici/julia_1.2.0:ubuntu18*

2. Start 2 terminal windows
3. At one terminal, and in a folder of your choice, clone the building emulator repository at Building Control Emulator , which also includes the script *runBuildingEmulatorDocker.sh* that allows you to start the building emulator docker as root.
4. At the other terminal, and in a folder of your choice, clone the adaptive MPC repository at Adaptive MPC , which also includes the *runMPCDocker.sh* that allows you to start adaptive MPC docker as root.
5. In the building emulator terminal, switch to */mnt/examples/* folder and run

```
python startREST.py -p ./models/LargeBuilding.fmu -s 60
```

6. In the Julia docker terminal, switch to inlineCode{/mnt/mcp} folder and run

```
julia simulate.jl
```

7. **WARNING!** I believe that Sen changed the *wrapped.fmu* model in terms of signals being communicated and their names, which implies that the MPC code would have to be, once again, changed. Needs to be checked if we want to use that model.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index