Enhanced OpenModelica Python Interface

by

Sudeep Bajracharya

LIU-IDA/LITH-EX-A--16/014—SE

2016-04-05
Final Thesis

Enhanced OpenModelica Python Interface

by

Sudeep Bajracharya

LIU-IDA/LITH-EX-A--16/014—SE

2016-04-05

Supervisor: Lena Buffoni
Examiner: Peter Fritzon
OMPython is a Python library for OpenModelica, which provides a Python interface to OpenModelica. This thesis significantly improves OMPython by an enhanced, more powerful and easier to use API. It presents how a user can use the Python interface to simulate and access Modelica models using Python objects. The enhanced OMPython includes the list of functions that have been implemented such as getXXXNames(), getXXXValues(), setXXXValues(), getXXXOptions(), setXXXOptions(), and simulate(), etc. that allow users to interact with Modelica model properties (i.e., model variables) and the output of the OpenModelica compiler and simulator. A few Modelica models are used for demonstrations in order to make it easy for readers to understand. By the way, this thesis does not describe modeling Modelica models or compilation of such models.
Acknowledgement

I feel myself very grateful to be part of this thesis, thank you Professor Peter Fritzson (IDA, SaS) for giving me an opportunity to be part of this project. I would like to thank personally a number of persons without whom it would not have been possible for me to complete this thesis. I would like to thank my supervisor Lena Buffoni (IDA, SaS, Postdok), Arun Kumar (IDA, SaS, Tekniker), Adeel Asghar (IDA, SaS, Tekniker), Martin Sjölund (IDA, SaS, Forskningsingerjör) and Alachew Mengist (IDA, SaS, Doktorand), who assisted me in so many ways, from gaining knowledge of the system till the implementation phase. I am very thankful to PELAB administrator for providing me a room where I could do my work without any disturbance. I could not skip my wife (Rosy), who encouraged me throughout this project. Last but not the least, special thanks to Professor Bernt Lie (University College of Southeast Norway), who proposed this thesis and helped in planning and designing sections too.
# Table of content

1 Introduction  
---
1.1 Motivation  
---
1.2 Purpose  
---
1.3 Goal  
---

2 Background Study  
---
2.1 Modelica  
---
2.2 OpenModelica  
---
2.3 OMPython  
---
2.4 OMParse  
---
2.5 FMU  
---
2.6 Optimization  
---
2.7 Linearization  
---
2.8 Case Studies  
---

3 Methodology  
---
3.1 Description  
---
3.2 Use-Case Diagram and Work Flow  
---

4 Implementation  
---
4.1 Initialization  
---
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>Display Details of quantities</td>
<td>15</td>
</tr>
<tr>
<td>4.3</td>
<td>Continuous quantity</td>
<td>16</td>
</tr>
<tr>
<td>4.4</td>
<td>Parameter Quantity</td>
<td>17</td>
</tr>
<tr>
<td>4.5</td>
<td>Input Quantity</td>
<td>18</td>
</tr>
<tr>
<td>4.6</td>
<td>Output Quantity</td>
<td>19</td>
</tr>
<tr>
<td>4.7</td>
<td>Simulation</td>
<td>19</td>
</tr>
<tr>
<td>4.8</td>
<td>Extract Result</td>
<td>20</td>
</tr>
<tr>
<td>4.9</td>
<td>FMU</td>
<td>21</td>
</tr>
<tr>
<td>4.10</td>
<td>Optimization</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>Linearization</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>Result and Test</td>
<td>24</td>
</tr>
<tr>
<td>5.1</td>
<td>Result</td>
<td>24</td>
</tr>
<tr>
<td>5.2</td>
<td>Test</td>
<td>35</td>
</tr>
<tr>
<td>6</td>
<td>Issues and Limitation</td>
<td>48</td>
</tr>
<tr>
<td>6.1</td>
<td>Regarding System</td>
<td>48</td>
</tr>
<tr>
<td>6.2</td>
<td>Regarding Application</td>
<td>48</td>
</tr>
<tr>
<td>7</td>
<td>Future Work</td>
<td>49</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions</td>
<td>50</td>
</tr>
<tr>
<td>9</td>
<td>Appendix</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>Tools used</td>
<td>51</td>
</tr>
</tbody>
</table>

IV Enhanced OpenModelica Python Interface
List of Figures

Figure 2-1 Architecture of OpenModelica ................................................................. 2
Figure 2-2 Python - OpenModelica Interface ............................................................ 3
Figure 2-3 OpenModelica Connection Editor ......................................................... 6
Figure 2-4 Simulation Result of the model ............................................................... 7
Figure 2-5 Plot Result of "time" Vs "h" ................................................................. 8
Figure 3-1 Agile Implementation Methodology ...................................................... 10
Figure 3-2 UCD of ModelicaSystem ................................................................. 11
Figure 3-3 Work Flow of ModelicaSystem ......................................................... 12
List of Tables

Table 2-1 VDP.mo ......................................................................................................................... 9
Table 2-2 Simulation using JModelica............................................................................................ 9
Table 4-1 BouncingBall.mo ........................................................................................................... 13
Table 4-2 dcmotor.mo ................................................................................................................... 14
Table 5-1 FractionCO2Room.mo .................................................................................................. 24
Table 5-2 Resulted linear_diesel_model.mo .................................................................................. 44
Table 5-3 Resulted linear_BouncingBall.mo .................................................................................. 46
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMC</td>
<td>OpenModelica Compiler</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>ORB</td>
<td>Object Request Broker</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>FMU</td>
<td>Functional Mock-up Unit</td>
</tr>
<tr>
<td>FMI</td>
<td>Functional Mock-up Interface</td>
</tr>
<tr>
<td>OCP</td>
<td>Optimal Control Problems</td>
</tr>
<tr>
<td>OSMC</td>
<td>Open Source Modelica Consortium</td>
</tr>
<tr>
<td>OM</td>
<td>OpenModelica</td>
</tr>
<tr>
<td>OMEdit</td>
<td>OpenModelica Connection Editor</td>
</tr>
<tr>
<td>OMPython</td>
<td>OpenModelica Python</td>
</tr>
<tr>
<td>OMAParser</td>
<td>OpenModelica Parser</td>
</tr>
<tr>
<td>UCD</td>
<td>User-Case Diagram</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Motivation

MATLAB is a high-performance language for technical computing. It is a favorable computational tool in many engineering fields such as control engineering, computational chemical and mechanical engineering, etc. Simulink, which is a graphical editor, customizable block libraries and solvers for modeling and simulating dynamic systems, is well integrated with MATLAB.

The disadvantage of MATLAB is that it is expensive (not free even for academic purpose), MATLAB as a programming language, is not ideally constructed and it is not well suited for describing complex models with recirculation, etc.

Modelica on the other hand, an objected-oriented declarative language, could replace Simulink. But, it lacks the richness of MATLAB in the context of functionality through toolboxes such as a control toolbox, an optimization toolbox, system identification toolbox, statistics toolbox, GUI building toolbox, etc. [1]

Nevertheless, Modelica together with Python can fulfill those features, perhaps not at the same level and quality as MATLAB, but to some useful level. Thus, Modelica and Python could be an alternative to MATLAB to some extent.

1.2 Purpose

The purpose of this thesis is to create a Python object that can interact with OMC and extract its results and let user do necessary data manipulation with the results, thus providing an alternative to MATLAB.

1.3 Goal

The main goal of this thesis is to:

- Create an object oriented environment for handling the OpenModelica APIs
- Implement methods that will interact with the OMC's result
- Make environment as convenient as possible from the usability point of view
2 Background Study

2.1 Modelica

Modelica is free, object-oriented modeling language. In short Modelica is a non-proprietary, equation-based language that can model conveniently complex physical systems containing, e.g., mechanical, electrical, electronic, thermal and process-oriented subcomponents, etc. [2]

2.2 OpenModelica

OpenModelica is one of the environments for creating and simulating models (equation based) which can be used for both industrial and academic purposes. The OM is supported by a non-profit organization called OSMC.

![Figure 2-1 Architecture of OpenModelica](image-url)
The figure 1-1 depicts the architecture of OM. It consists of several subsystems. OM is facilitated by number of editors as shown in the figure 1-1. Data and control flow of the system is represented by arrows. From an editor, command is sent to the interactive session handler. The command is then compiled and executed by respective subsystems and returns the result of the corresponding command.

Among many, one of the features of OpenModelica is, it provides OM Connection Editor which is advanced open source user friendly front-end graphical interface. In the OM Connection Editor, user can create a model of simple systems to a complex system. There are several others features including connection editing, simulation of models, plotting result, representation of models in both textual (coding) and graphical form. [3, 4]

Basically, this thesis is based on creating methods to interact with the models results rather than creating models, so it focuses more on back-end part for accessing a model. The tutorial for modeling using OpenModelica can be found in URL http://openmodelica.org

![Figure 2-2 Python - OpenModelica Interface](image-url)
2.3 OMPython

Python interface to OpenModelica, is a free, open source, highly portable Python based interactive session handler for Modelica scripting. It provides an environment for user to load, compile and simulate models based on the OpenModelica library standard available. OMPython is one of the packages or modules of OpenModelica rather than a separate single package. OMPython is included in OpenModelica installation. It is implemented in Python using the OmniORB and OmniORBpy which is high performance CORBA, to interact with OMC. [5]

Features of OMPython:

- Creation of Modelica models, simulation, plotting results.
- Supports all OpenModelica API calls
- Optimized parser results for easy access of OMC’s output, etc

2.4 OMParse

OMParser is a sub-module of OMPython. It is one of the important components of OMPython as it plays vital role for an interaction between Python and OpenModelica. When there is the interaction between Python and OpenModelica via CORBA, it will be in string-to-string format which is not appropriate way to fetch results from OMC all the time. So, OMParse parses the output into proper data structures. [5]

Features of OMParse are to:

- Analyze the CORBA strings for categorical data
- Group each category under a category name
- Typecast the data within these categories
- Build suitable data structures to hold these data so that the results can be easily accessed
- Return the results to the OMPython module

2.5 FMU

A FMI is a standard for model exchange and co-simulation of dynamic system models. The pre-compiled models i.e., C-code, binaries and description files are packed in a FMU. FMI exports models in three different ways;

1. Model Exchange
2. Co-Simulation
3. Model Exchange and Co-Simulation

There are two versions of FMI: version 1.0 and 2.0. The OM supports both versions. By default, in this system, to convert FMU to Modelica models, it sets FMU type as model exchange and version as 1.0. [4]

2.6 Optimization

OM provides built-in dynamic optimization of models that allow users to define OCP using Modelica language and optimization language extension called “Optimica”. [4]

2.7 Linearization

Library Modelica_LinearSystem2 is a Modelica Package that provides different representations of linear, time invariant differential equation system in state space form: [6]

\[
\begin{align*}
der (x) &= A \ast x + B \ast u \\
y &= C \ast x + D \ast u
\end{align*}
\]

2.8 Case Studies

In order to begin this thesis, we did few case studies. At first installed OpenModelica and became familiar with the architecture of OpenModelica. Then got familiarized with OpenModelica Connection Editor, basic modeling language, how to load Modelica models, simulating models, plotting results and so on. After that, analyzed OMPython bundle, gained the basic knowledge about interaction between Python and OMC.
The pictorial representation of OMEdit is shown in figure 2-3 below. The user can easily write model as shown in the figure. There are several built-in libraries where user can use directly if needed in their models.

![Figure 2-3 OpenModelica Connection Editor](image)

```
1 model BouncingBall
2 parameter Real e = 0.7 "coefficient of restitution";
3 parameter Real g = 9.81 "gravity acceleration";
4 Real h(start = 2) "height of ball";
5 Real v "velocity of ball";
6 Boolean flying(start = true) "true, if ball is flying";
7 Boolean impact;
8 Real v_new;
9 Integer foo;
10 equation
11 impact = h <= 0;
12 foo = if impact then 1 else 2;
13 dash(v) = if flying then -g else 0;
14 dash(h) = v;
15 when (h <= 0.0 and v <= 0.0, impact) then
16 v_new = if edge(impact) then -e * pre(v) else 0;
17 flying = v_new > 0;
18 reinit(v, v_new);
19 end when;
20 end BouncingBall;
```
The model can be easily simulated either from "Simulation" menu or directly from the Simulation toolbar. Beside, user can set the simulation properties and/or simulate with debugging mode, etc. The following figure 2-4 depicts the simulation result of the model (BouncingBall.mo).

![Simulation Output](image)

**Figure 2-4 Simulation Result of the model**
Once the simulation is completed successfully, user can plot the result of simulation. The following figure 2-5, shows the plot result of time Vs height ('time' Vs 'h').

![Figure 2-5 Plot Result of "time" Vs "h"](image)

In the mean time, we did basic study on another Modelica-based open source platform called “JModelica”. JModelica is used to: [7]

- Model a system using Modelica
- Solve a complex simulation and optimization problems using numerical algorithms
- Automate using Python scripting environment
- Visualize results

We installed JModelica and tested few Modelica models by simulating Modelica models using Python. It helped us to gain some knowledge and gave an idea to build a road-map. In fact, the concept of creating an object oriented Python to interact with OMC output has been inspired from JModelica.
A simple example to simulate Modelica model called "VDP.mo" (from JModelica user documentation), in table 2-1 below, using JModelica is shown below. [8]

Table 2-1 VDP.mo

```model VDP
  // State start values
  parameter Real x1_0 = 0;
  parameter Real x2_0 = 1;
  // The states
  Real x1(start = x1_0);
  Real x2(start = x2_0);
  // The control signal
  input Real u;
  equation
  der(x1) = (1 - x2^2) * x1 - x2 + u;
  der(x2) = x1;
end VDP;
```

First, open Python shell. Then import the compile_fmu and load_fmu. Then compile and load the model. Finally, simulate the model as shown in table 2-2 below:

Table 2-2 Simulation using JModelica

```python
# Import the function for compilation of models and the load_fmu method
from pymodelica import compile_fmu
from pyfmi import load_fmu
# Compile model
fmu_name = compile_fmu("VDP","VDP.mo")
# Load model
vdp = load_fmu(fmu_name)
# Simulate model
res = vdp.simulate(final_time=10)
```

(More tutorials about JModelica is available at www.jmodelica.org)
3 Methodology

3.1 Description

In this project, as a hard and fast rule, not any particular methodology is being used. But in general, it follows more or less agile methodology. The project is developed in iterative approach. In each iteration, a module (function/method) is incremented in terms of performance. The following figure 3-1 depicts the implementation process for each module.

Figure 3-1 Agile Implementation Methodology
During the implementation phase, there were several Skype meetings and communication via emails with Professor Bernt Lie. In each meeting, there were discussions about results of modules. Besides, there were several meetings with project assistants where results were discussed and also planning were done for new modules.

For each module, it was planned first, and then implemented codes for corresponding module. Simultaneously, executed the codes and tested the results as shown in the figure 3-1. Precisely, the results were presented to Professor Bernt Lie or project assistants. They gave feedbacks and comments regarding the results. The process was repeated again and again for each module until there were not expected results.

3.2 Use-Case Diagram and Work Flow

![Figure 3-2 UCD of ModelicaSystem](image-url)
The above figures 3-2 and 3-3 represent the UCD and work flow respectively for this system. In order to use the object oriented OMPython, firstly, a user must import the "ModelicaSystem" from "OMPython" module. Then create an object for a model. This loads and builds the model, meaning it will generate a number of corresponding files such as object, source, json, and xml files, etc. User can get/set required quantities (i.e. model variables). Then simulate the model (before simulation, user could also set simulation options). Extract the simulation result, plot the result and also do other data manipulation as per requirement. Besides, a user can also optimized a model and/or create a corresponding FMU and/or linearized a model.
4 Implementation

The main goal is to create an environment for the users so that it would become easy to simulate the model, check the results of simulation, and plot the required results and so on. Above all, to present an object oriented environment where the user can create an object and do further necessary operations with the result obtained from the simulation. Additionally, to facilitate user to get/set values of quantities. Indeed in model, there might be some quantities whose value cannot be changed, thus for those quantities, values could not be set.

So in order to provide this kind of platform, we have design a Python class whose object has capability to store the results of simulation from the OMC and then do the necessary data manipulation.

Basically, for this, we have created an object oriented class called “ModelicaSystem” in OMPython module of OpenModelica package. All the codes are written inside the class ModelicaSystem. By the help of OMPython (CORBA, OMParse, OpenModelica APIs), the methods that have been implemented, are able to simulate Modelica model and access/manipulate the result/output of OMC.

In the following sections, there is the description about implemented methods. In order to make an example usage clearer, we have used some of the Modelica models where the models are relevant. We have used "BouncingBall.mo" as shown in table 4-1 and "dcmotor.mo" as shown in table 4-2, which is located in the directory of OpenModelica at "C:\OpenModelica1.9.4-dev.beta2\share\doc\omc\testmodels".

Table 4-1 BouncingBall.mo

```
model BouncingBall
    parameter Real e = 0.7 "coefficient of restitution";
    parameter Real g = 9.81 "gravity acceleration";
    Real h(start = 2) "height of ball";
    Real v "velocity of ball";
    Boolean flying(start = true) "true, if ball is flying";
    Boolean impact;
    Real v_new;
    Integer foo;
    equation
        impact = h <= 0.0;
        foo = if impact then 1 else 2;
        der(v) = if flying then -g else 0;
```
der(h) = v;
when {h <= 0.0 and v <= 0.0, impact} then
  v_new = if edge(impact) then -e * pre(v) else 0;
  flying = v_new > 0;
  reinit(v, v_new);
end when;
end BouncingBall;

4.1 Initialization

4.1.1 ModelicaSystem()

It is a constructor that initializes to load a file and build a model that generates object, exe, xml and json files, etc. It can be called:

- without any arguments: In this case it neither loads a file nor build a model. This is useful when a FMU needed to convert to Modelica model (section 4.9)
- with two arguments as file name with ".mo" extension and the model name respectively
- with three arguments, the first and second are file name and model name respectively and the third arguments is Modelica standard library to load a model, which is common in such models where the model is based on the standard library. For example, here is a model named "dcmotor.mo" below table 4-2, which is located in the directory of OpenModelica at "C:\OpenModelica1.9.4-dev.beta2\share\doc\omc\testmodels".

**Note:** If a model file is not in the current working directory, then the path where the file is located must be included together with the file name. Besides, if the Modelica model contains several different models within the same package, then in order to build the specific model, in second argument, user must put the package name with dot(.) followed by specific model name.

Example usage:

```python
mod = ModelicaSystem()
mod = ModelicaSystem("BouncingBall.mo","BouncingBall")
mod = ModelicaSystem("dcmotor.mo","dcmotor","Modelica")
mod = ModelicaSystem("C:/myModels/BouncingBall.mo","BouncingBall")
mod = ModelicaSystem("FractionCO2Room.mo","FractionCO2Room.ModFractionCO2Room")
```

Table 4-2 dcmotor.mo

| model dcmotor |
|---|---|
| Modelica.Electrical.Analog.Basic.Resistor resistor1(R = 10); |
//Observe the difference between MSL 2.2 and 3.1 regarding the default values, in 3.1 there are no default values set, only start values
Modelica.Electrical.Analog.Basic.Inductor inductor1(L = 0.2);
Modelica.Electrical.Analog.Basic.Ground ground1;
Modelica.Mechanics.Rotational.Components.Inertia load(J = 1);  // Modelica version 3.1
// Modelica.Mechanics.Rotational.Inertia load(J = 1);  // Modelica version 2.2
Modelica.Electrical.Analog.Basic.EMF emf1;
Modelica.Blocks.Sources.Step step1;
Modelica.Electrical.Analog.Sources.SignalVoltage signalVoltage1;
equation
//connect(step1.outport, signalVoltage1.inPort);
connect(step1.y, signalVoltage1.v);
connect(signalVoltage1.p, resistor1.p);
connect(resistor1.n, inductor1.p);
connect(inductor1.n, emf1.p);
// connect(emf1.flange_b, load.flange_a);  //Modelica version 2.2
connect(emf1.flange, load.flange_a);  // Modelica version 3.1
connect(signalVoltage1.n, ground1.p);
connect(ground1.p, emf1.n);
end dcmotor;

4.2 Display Details of quantities

4.2.1 getQuantities()

This method returns list of dictionaries. It displays details of quantities such as name, value, changeable, and description, where changeable means if value for corresponding quantity name is changeable or not. It can be called:

- without argument: it returns list of dictionaries of all quantities
- with a single argument as list of quantities name in string format: it returns list of dictionaries of only particular quantities name
- a single argument as a single quantity name (or in list) in string format: it returns list of dictionaries of the particular quantity name

Example usage:

```python
mod.getQuantities()
mod.getQuantities(['h', 'flying']) #BouncingBall.mo
mod.getQuantities('flying')  #which results same as mod.getQuantities(['flying'])
```
4.3 Continuous quantity

4.3.1 getContinuousNames()

This method returns list of quantities name that are continuous. It can be called:

- only without any arguments: returns the list of quantities (continuous) names

Example usage:

```python
conti_names = mod.getContinuousNames()
```

4.3.2 getContinuousValues()

This method returns list of values of the quantities name that are continuous. It can be called:

- without any arguments: returns list of values of all quantities name that are continuous
- with a single argument as continuous name in string format: returns value of the corresponding name
- with a single argument as list of continuous names in string format: return list of values of the corresponding names.

1. If the list of names is more than one and it is being assigned by single variable then it returns the list of values of the corresponding names.
2. If the list of names is more than one and it is being assigned by same number of variable as the number of element in the list then it will return the value to the variables correspondingly (Python unpacking)

Example usage:

```python
conti_values = mod.getContinuousValues()
contiV_value = mod.getContinuousValues('h') #return float
contiV_value = mod.getContinuousValues(['h']) #return value in list
contiV_value = mod.getContinuousValues(['h','v']) # return list of values
Val1, Val2 = mod.getContinuousValues(['h', 'v']) #Val1 = value-of-'h' and Val2 = value-of-'v'
```

4.3.3 setContinuousValues()

This method sets values for the quantities name that are continuous. It can be called:

- with only two (list) arguments, list of quantities names(in string format) and list of values to be set respectively: set the values for the corresponding name
Example usage:
```
mod.setContinuousValues(['h', 'v'],[3.0, 2.5])
```

## 4.4 Parameter Quantity

### 4.4.1 getParameterNames()

This method returns list of quantities name that are parameters. It can be called:

- only without any arguments: returns list of quantities (parameter) name

Example usage:
```
para_names = mod.getParameterNames()
```

### 4.4.2 getParameterValues()

This method returns list of values of the quantities name that are parameters. It can be called:

- without any arguments: return list of values of all quantities (parameter) name
- with a single argument as parameter name in string format: returns value of the corresponding name
- with a single argument as list of parameter names in string format: return list of values of the corresponding names.

1. If the list of names is more than one and it is being assigned by single variable then it returns the list of values of the corresponding names
2. If the list of names is more than one and it is being assigned by same number of variable as the number of element in the list then it will return the value to the variables correspondingly (Python unpacking)

Example usage:
```
para_values = mod.getParameterValues()
para_value = mod.getParameterValues('g') #return float
para_value = mod.getParameterValues(['g']) #return value in list
para_value = mod.getParameterValues(['e','g']) #return list of values
pv1, pv2 = mod.getParameterValues(['e','g']) #pv1=value-of-'e' and pv1=value-of-'g'
```
4.4.3 setParameterValues()

This method sets values for the quantities name that are parameters. It can be called:

- only with two (lists) arguments, list of quantities names (in string format) and list of values correspondingly: set the values for the corresponding names

Example usage:

```python
mod.setParameterValues(['e'],[0.9]) #BouncingBall.mo
```

4.5 Input Quantity

4.5.1 getInputNames()

This method returns list of quantities name that are inputs. It can be called:

- only without any arguments: returns the list of quantities (input) name

Example usage:

```python
input_names = mod.getInputNames()
```

4.5.2 getInputValues()

This method returns list of values of the quantities name that are inputs. It can be called:

- without any arguments: returns list of values of all quantities (input) name
- with a single argument as input name in string format: returns list of values of the corresponding name

Example usage:

```python
input_values = mod.getInputValues()
input_values=mod.getInputValues('N') #ref: FractionCO2Room
```

4.5.3 setInputValues()

This method sets values for the quantities name that are inputs. It can be called:

- with only two arguments, input name in string format and list of values for the corresponding input. The list should be in tuple form e.g. [(t1,v1), (t2,v2), ..., (tN,vN)] where t1 <= t2 <= ... <= tN, and where a straight line is assumed between each temporal value (tj) and input value (vj)
4.6 Output Quantity

4.6.1 getOutputNames()

This method returns list of quantities name that are outputs. It can be called:

- only without any arguments: returns the list of all quantities (output) name

Example usage:

```python
Output_names = mod.getOutputNames()
```

4.6.2 getOutputValues()

This method returns list of values of the quantities name that are outputs. It can be called:

- only without any arguments: returns the list of values of all output name

Example usage:

```python
output_values = mod.getOutputValues()
```

Note: Test has not been carried out for Output quantities due to the lack of model that contains output.

4.7 Simulation

4.7.1 getSimulationOptions()

This method displays the simulation options name with corresponding options values. It can be called:

- only without any arguments: displays simulation options name with corresponding values

The simulation options are set by default, but the options can be changed as per user requirement.

Example usage:
4.7.2 setSimulationOptions()

This method is used to set simulation options. It can be called:

- with a sequence of simulation options name and assigning corresponding values as arguments as show in the example usage below.

Example usage:

```python
mod.setSimulationOptions(stopTime = 100, solver = 'euler') #BouncingBall.mo
```

4.7.3 simulate()

This method simulates model according to the simulation options. It can be called:

- only without any arguments: simulates the model

Example usage:

```python
mod.simulate()
```

4.8 Extract Result

4.8.1 getSolutions()

This method returns tuple of numpy arrays. It can be called:

- with a list of quantities name in string format as argument: it returns the simulation results of the corresponding names in the same order. Here it supports Python unpacking depending upon the number of variables assigned.

Example usage:

```python
time, height = mod.getSolutions(['time', 'h']) #BouncingBall.mo
```
4.9 FMU

4.9.1 convertMo2Fmu()
This method is used to generate FMU from the given Modelica model. It creates "modelName.fmu" in the current working directory. It can be called:
- only without any arguments

Example usage:
```
mod.convertMo2Fmu()
```

4.9.2 convertFmu2Mo()
In order to load FMU, at first it needs to be translated into Modelica model. This method is used to generate Modelica model from the given FMU. It generates "fmuName_me_FMU.mo". It can be called:
- only without any arguments

Currently, it only supports Model Exchange conversion.

Example usage:
```
mod.convertFmu2Mo("fmuFile.fmu")
```

4.10 Optimization

4.10.1 getOptimizationOptions()
This method displays the optimization options name and its corresponding value. Initially, values are set by default. It can be called:
- only without any arguments

Example usage:
```
mod.getOptimiationOptions()
```

4.10.2 setOptimizationOptions()
This method is used to set optimization options. It can be called:
• with a sequence of optimization options name and assigning corresponding values as arguments as show in the example usage below

Example usage:
```
mod.setOptimizationOptions(stopTime = 10, simflags = '-lv LOG_IPOPT -optimizerNP 1')
```

### 4.10.3 optimize()

This method optimizes model according to the optimized options. It can be called:

• only without any arguments

Example usage:
```
mod.optimize()
```

### 4.11 Linearization

#### 4.11.1 getLinearizationOptions()

This method displays the linearization options name with corresponding options value. It can be called:

• only without any arguments

The linearization options are set by default, but the options can be changed as per user requirement.

Example usage:
```
mod.getLinearizationOptions()
```

#### 4.11.2 setLinearizationOptions()

This method is used to set linearization options. It can be called:

• with a sequence of linearization options name and assigning corresponding value as arguments as show in the example usage below

Example usage:
```
mod.setLinearizationOptions(stopTime=0, stepSize=10)
```
4.11.3 linearize()

This method linearizes model according to the linearized options. This will generate a linear model that consists of matrices A, B, C and D. It can be called:

- only without any arguments

Example usage:

```python
mod.linearize()
```
5 Result and Test

5.1 Result

We would like to add the result of simulation of the model named "FractionCO2Room.mo" as shown in table 5-1, which is provided by Professor Bernt Lie. In this model FractionCO2Room, there are three models SimFractionCO2Room, ModFractionCO2Room and ModFractionCO2RoomODE. At first, the model SimFractionCO2Room is executed and then secondly, ModFractionCO2Room is executed. The model is executed in Jupyter NoteBook, web application. The screen shots are presented as the result of execution.

<table>
<thead>
<tr>
<th>Table 5-1 FractionCO2Room.mo</th>
</tr>
</thead>
<tbody>
<tr>
<td>package FractionCO2Room</td>
</tr>
<tr>
<td>// Package for simulating CO2 fraction in non-isothermal room</td>
</tr>
<tr>
<td>// author: Bernt Lie</td>
</tr>
<tr>
<td>model SimFractionCO2Room</td>
</tr>
<tr>
<td>// Simulation of CO2 fraction in non-isothermal room</td>
</tr>
<tr>
<td>// author: Bernt Lie</td>
</tr>
<tr>
<td>// Telemark University College</td>
</tr>
<tr>
<td>// September 17, 2015</td>
</tr>
<tr>
<td>// Revised: November 12, 2015</td>
</tr>
<tr>
<td>// Revised: November 17, 2015</td>
</tr>
<tr>
<td>// Instantiate model of <em>fraction of CO2</em> (fCO2)</td>
</tr>
<tr>
<td>ModFractionCO2Room fCO2;</td>
</tr>
<tr>
<td>ModFractionCO2RoomODE fCO2ODE;</td>
</tr>
<tr>
<td>// Declaring variables</td>
</tr>
<tr>
<td>// -- inputs</td>
</tr>
<tr>
<td>Real _Vdi &quot;Volumetric rate, L/s&quot;;</td>
</tr>
<tr>
<td>Real _xCO2i &quot;Influent CO2 fraction, -&quot;;</td>
</tr>
<tr>
<td>Integer _N &quot;Number of persons in room, -&quot;;</td>
</tr>
<tr>
<td>Real _Ti &quot;Influent temperature, K&quot;;</td>
</tr>
<tr>
<td>Real _Qdi &quot;Heating, kW&quot;;</td>
</tr>
<tr>
<td>// -- outputs</td>
</tr>
<tr>
<td>output Real _xCO2ppm &quot;Mole fraction CO2 in classroom, -&quot;;</td>
</tr>
<tr>
<td>output Real _xCO2ippm &quot;Influent mole fraction CO2, -&quot;;</td>
</tr>
<tr>
<td>output Real _TC &quot;Temperature in room, C&quot;;</td>
</tr>
</tbody>
</table>
output Real _TiC "Influent temperature, C"

output Real _QdkW "Total heating, kW";
output Real _QdgkW "Heating from persons, kW";
output Real _QdikW "Heating, kW";
output Real _delHdvkW "Heat loss by ventilation, kW";

//
output Real _xCO2ODEppm "Mole fraction CO2 in classroom from ODE, ppm";
output Real _TCODE "Temperature in room from ODE, C";
// Equations
equation

// -- input values
_Vdi = if time < 8000 then 4 * fCO2.V / 3600 else 6 * fCO2.V / 3600;
_xCO2i = if time < 16000 then 400e-6 else 600e-6;
_N = if mod(time, 3600) < 2700 then 15 else 5;
_Ti = if time < 24000 then 273.15 + 10 else 273.15;
_Qdi = 0;
// -- injecting input functions to model inputs
fCO2.Vdi = _Vdi;
fCO2.xCO2i = _xCO2i;
fCO2.N = _N;
fCO2.Ti = _Ti;
fCO2.Qdi = _Qdi;
//
fCO2ODE.Vdi = _Vdi;
fCO2ODE.xCO2i = _xCO2i;
fCO2ODE.N = _N;
fCO2ODE.Ti = _Ti;
fCO2ODE.Qdi = _Qdi;
// -- outputs
_xCO2ppm = fCO2.xCO2 * 1e6;
_xCO2ippm = fCO2.xCO2i * 1e6;
_TC = fCO2.T - 273.15;
_TiC = fCO2.Ti - 273.15;
_QdkW = fCO2.Qd / 1e3;
_QdgkW = fCO2.Qdg / 1e3;
_QdikW = fCO2.Qdi / 1e3;
_delHdvkW = (fCO2.Hdi - fCO2.Hde) / 1e3;
//
_xCO2ODEppm = fCO2ODE.xCO2 * 1e6;
_TCODE = fCO2ODE.T - 273.15;
end SimFractionCO2Room;

//

model ModFractionCO2Room
// Model of CO2 fraction in non-isothermal room
// author: Bernt Lie
// Telemark University College
// September 17, 2015
// Revised: November 12, 2015
//
// Constant
constant Real R = 8.31e3 "Gas constant, J/(kmol.K)";
// Parameters
parameter Real V = 45 * 3 "Classroom volume, m3";
parameter Real Mair = 29 "Molar mass of air, kg/kmol";
parameter Real Ke = 100 / 1.01e5 "Effluent valve constant, kg/(s.Pa)";
parameter Real pa = 1.01e5 "Atmospheric pressure, Pa";
parameter Real MCO2 = 12 + 2 * 16 "Molar mass of CO2, kg/kmol";
parameter Real mdCO2g0 = 1 / (24 * 3600) "CO2 breathing per person, kg/(s.#persons)";
parameter Real Qdg0 = 100 "Heat generated per person, W";
parameter Realetatav = 0.6 "Heat recovery factor, -";
// Initial state parameters
parameter Real p0 = pa "Initial pressure in classroom, Pa";
parameter Real m0 = p0 * V * Mair / (R * T0) "Initial mass in classroom, kg";
parameter Real nCO20 = V * xCO20 * p0 / (R * T0) "Initial number of CO2 moles, kmol";
parameter Real T0 = 293 "Initial temperature in classroom, K";
parameter Real Hh0 = FncHh(T0) "Initial specific enthalpy, kJ/kg";
parameter Real U0 = Hh0 * m0 - p0 * V "Initial internal energy, kJ";
// Declaring variables
// -- states
Real m(start = m0, fixed = true) "Mass of air in classroom, kg";
Real nCO2(start = nCO20, fixed = true) "Number of moles CO2 in classroom, kmol";
Real U(start = U0, fixed = true) "Internal energy, kJ";
// -- auxiliary variables
Real mdi "Influent mass flow rate of air, kg/s";
Real mde "Effluent mass flow rate of air, kg/s";
Real ndCO2i "Influent molar flow rate of CO2, kmol/s";
Real ndCO2e "Effluent molar flow rate of CO2, kmol/s";
Real ndCO2g "Molar rate of generation of CO2, kmol/s";
Real p "Total pressure in classroom, Pa";
Real n "Total number of moles in classroom, kmol";
Real ndi "Influent total number of moles to classroom, kmol/s";
Real xCO2 "Mole fraction of CO2 in classroom, -";
Real pCO2i "Influent partial pressure of CO2, Pa";
Real nde "Effluent molar flow rate, kmol/s";
Real mdCO2g "Mass of generation of CO2 through breathing, kg/s";
Real H "Total enthalpy, kJ";
Real Hh "Specific enthalpy, kJ/kg";
Real T "Temperature, K";
Real Hdi "Influent enthalpy flow, kW";
Real Hhi "Influent specific enthalpy, kW/kg";
Real Hde "Effluent enthalpy flow, kW";
Real Qd "Heat flow, kW";
Real Qdg "Heat generated by people, kW";

// -- input variables
input Real Vdi "Influent volumetric flow rate, m3/s";
input Real xCO2i "Influent mole fraction of CO2, -";
input Integer N "Number of persons in classroom, -";
input Real Ti "Influent temperature -- ambient temperature, K";
input Real Qdi "Heating, kW";

// Equations constituting the model

equation

// Differential equations
der(m) = mdi - mde;
der(nCO2) = ndCO2i - ndCO2e + ndCO2g;
der(U) = Hdi - Hde + Qd;

// Algebraic equations
m = n * Mair;
pa * Vdi = ndi * R * Ti;
mdi = ndi * Mair;
mde = Ke * (p - pa);
p * V = n * R * T;

//
pCO2i = xCO2i * pa;
pCO2i * Vdi = ndCO2i * R * Ti;
nde = mde / Mair;
xCO2 = nCO2 / n;
ndCO2e = xCO2 * nde;
ndCO2g = mdCO2g / MCO2;
mdCO2g = mdCO2g0 * N;
//
U = H - p * V;
H = m * Hh;
Hh = FncHh(T);
Hhi = FncHh(Ti);
Hdi = mdi * (Hhi + etav * (Hh - Hhi));
Hde = mde * Hh;
Qd = Qdi + Qdg;
Qdg = Qdg0 * N;
end ModFractionCO2Room;

//

model ModFractionCO2RoomODE
// Model in ODE form of CO2 fraction in non-isothermal room
// author: Bernt Lie
// Telemark University College
// November 17, 2015
//
// Constant
custom Real R = 8.31e3 "Gas constant, J/(kmol.K)";
// Parameters
parameter Real V = 45 * 3 "Classroom volume, m3";
parameter Real Mair = 29 "Molar mass of air, kg/kmol";
parameter Real Ke = 100 / 1.01e5 "Effluent valve constant, kg/(s.Pa)";
parameter Real pa = 1.01e5 "Atmospheric pressure, Pa";
parameter Real MCO2 = 12 + 2 * 16 "Molar mass of CO2, kg/kmol";
parameter Real mdCO2g0 = 1 / (24 * 3600) "CO2 breathing per person, kg/(s.#persons)";
parameter Real Qdg0 = 100 "Heat generated per person, W";
parameter Real chp = 1e3 "Specific heat capacity, J/(kg.K)";
parameter Real etav = 0.6 "Heat recovery factor, -";
// Initial state parameters
parameter Real p0 = pa "Initial pressure in classroom, Pa";
parameter Real m0 = p0 * V * Mair / (R * T0) "Initial mass in classroom, kg";
parameter Real xCO20 = 400e-6 "Initial CO2 fraction in classroom, -";
// Declaring variables
// -- states
Real m(start = m0, fixed = true) "Mass of air in classroom, kg";
Real xCO2(start = xCO20, fixed = true) "Mole fraction CO2 in classroom, -";
Real T(start = T0, fixed = true) "Temperature in room, K";
// -- input variables
input Real Vdi "Influent volumetric flow rate, m3/s";
input Real xCO2i "Influent mole fraction of CO2, -";
input Integer N "Number of persons in classroom, -";
input Real Ti "Influent temperature -- ambient temperature, K";
input Real Qdi "Heating, kW";

// Equations constituting the model

// Differential equations
der(m) = pa * Mair * Vdi / (R * Ti) - Ke * (R * T * m / (V * Mair) - pa);
m * der(xCO2) = pa * Mair * Vdi / (R * Ti) * (xCO2i - xCO2) + Mair / MCO2 * mdCO2g0 * N;
m * (chp - R / Mair) * der(T) = (1 - etav) * pa * Mair * Vdi / (R * Ti) * chp * (Ti - T) + R * T / Mair * (pa * Mair * Vdi / (R * Ti) - Ke * (R * T * m / (V * Mair) - pa)) + Qdi + Qdg0 * N;
end ModFractionCO2RoomODE;

//

function FncHh "Specific enthalpy of air"
// Function for computing specific enthalpy of air
// author: Bernt Lie
// Telemark University College
// November 12, 2015
//
// Function input arguments
input Real T "Temperature, K";
// Function output (response) value
output Real Hh "Specific enthalpy";
// Local (protected) quantities
protected
parameter Real Hhdd = 0 "Standard state specific enthalpy, J/kg";
parameter Real Tdd = 298.15 "Standard state temperature, K";
parameter Real chp = 1e3 "Specific heat capacity, J/(kg.K)";
// Algorithm for computing specific enthalpy
algorithm
Hh := Hhdd + chp * (T - Tdd);
end FncHh;

// End package
end FractionCO2Room;
Execution of ModFractionCO2Room which is a system with inputs:

```python
import numpy.random as nr
import matplotlib.pyplot as plt
import pandas as p
from OMPython import ModelicaSystem

m2=ModelicaSystem("FractionCO2Room.mo","FractionCO2Room.ModFractionCO2Room")

frame = p.DataFrame(m2.getQuantities())

frame

<table>
<thead>
<tr>
<th>Changeable</th>
<th>Description</th>
<th>Name</th>
<th>Value</th>
<th>Variability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Internal energy, kJ</td>
<td>U</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>1</td>
<td>Mass of air in classroom, kg</td>
<td>m</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>2</td>
<td>Number of moles CO2 in classroom, kmol</td>
<td>nCO2</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>3</td>
<td>Internal energy, kJ</td>
<td>der(U)</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>4</td>
<td>Mass of air in classroom, kg</td>
<td>der(m)</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>5</td>
<td>Number of moles CO2 in classroom, kmol</td>
<td>der(nCO2)</td>
<td>None</td>
<td>continuous</td>
</tr>
</tbody>
</table>

frame2 = p.DataFrame(m2.getQuantities(['Ke', 'V']))

frame2

<table>
<thead>
<tr>
<th>Changeable</th>
<th>Description</th>
<th>Name</th>
<th>Value</th>
<th>Variability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Effluent valve constant, kg/(s.Pa)</td>
<td>Ke</td>
<td>0.0009900990099009901</td>
<td>parameter</td>
</tr>
<tr>
<td>1</td>
<td>Classroom volume, m3</td>
<td>V</td>
<td>135.0</td>
<td>parameter</td>
</tr>
</tbody>
</table>

paraName = m2.getParameterNames()
```
In [8]: paraName

Out[8]: ['Hh0', 'Ke', 'MCO2', 'Mair', 'Qdg0', 'T0', 'U0', 'V', 'etav', 'm0', 'mdCO2g0', 'nCO20', 'p0', 'pa', 'XC020']

In [9]: paraVal = m2.getParameterValues()

In [10]: paraVal

Out[10]: [None, 0.0009900990099009901, 44.0, 29.0, 100.0, 293.0, None, 135.0, 0.6, None, 1.157407407407407e-05, None, None, 101000.0, 0.0004]

In [11]: V = m2.getParameterValues('V')

In [12]: V

Out[12]: 135.0

In [13]: details_ = m2.getQuantities(paraName)
Enhanced OpenModelica Python Interface

```python
In [14]:
for d in details:
   display details without using a DataFrame
   print(d)

('Changeable': 'false', 'Description': ': Initial specific enthalpy, kJ/kg', 'Name': ': 'H0', 'Value': ': None, 'Variability': ': 'parameter')
('Changeable': 'true', 'Description': ': Effluent valve constant, kg/(s.Pa)', 'Name': ': 'Ke', 'Value': ': 0.0000000000000000081', 'Variability': ': 'parameter')
('Changeable': 'true', 'Description': ': Molar mass of CO2, kg/kmol', 'Name': ': 'MC02', 'Value': ': 44.0', 'Variability': ': 'parameter')
('Changeable': 'true', 'Description': ': Molar mass of air, kg/kmol', 'Name': ': 'Main', 'Value': ': 29.0', 'Variability': ': 'parameter')
('Changeable': 'true', 'Description': ': Heat generated per person, W', 'Name': ': 'Qdg0', 'Value': ': 100.0', 'Variability': ': 'parameter')
('Changeable': 'true', 'Description': ': Initial temperature in classroom, K', 'Name': ': 'T0', 'Value': ': 293.0', 'Variability': ': 'parameter')
('Changeable': 'false', 'Description': ': Initial internal energy, kJ', 'Name': ': 'U0', 'Value': ': None, 'Variability': ': 'parameter')
('Changeable': 'true', 'Description': ': Classroom volume, m3', 'Name': ': 'V', 'Value': ': 135.0', 'Variability': ': 'parameter')
('Changeable': 'true', 'Description': ': Heat recovery factor, -, 'Name': ': 'e_tav', 'Value': ': 0.6', 'Variability': ': 'parameter')
('Changeable': 'false', 'Description': ': Initial mass in classroom, kg', 'Name': ': 'm0', 'Value': ': None, 'Variability': ': 'parameter')
('Changeable': 'true', 'Description': ': CO2 breathing per person, kg/(s personnes)', 'Name': ': 'mdC02g0', 'Value': ': 1.157407407407407e-005', 'Variability': ': 'parameter')
('Changeable': 'false', 'Description': ': Initial number of CO2 moles, kmol', 'Name': ': 'nC020', 'Value': ': None, 'Variability': ': 'parameter')
('Changeable': 'false', 'Description': ': Initial pressure in classroom, Pa', 'Name': ': 'p0', 'Value': ': None, 'Variability': ': 'parameter')
('Changeable': 'true', 'Description': ': Atmospheric pressure, Pa', 'Name': ': 'pa', 'Value': ': 101300.0', 'Variability': ': 'parameter')
('Changeable': 'true', 'Description': ': Initial CO2 fraction in classroom, -', 'Name': ': 'xCO20', 'Value': ': 0.0004', 'Variability': ': 'parameter')

In [15]:
m2.getSimulationOptions()

startTime = 0.0
stopTime = 1.0
stepSize = 0.002
tolerance = 1e-006
solver = dassl

In [16]:
m2.setSimulationOptions(stopTime=7200, stepSize=10)

In [17]:
m2.getInputNames()

Out[17]: ['Qdi', 'Ti', 'Vdi', 'xCO2i', 'N']

In [18]:
m2.setInputValues("Qdi",[(0,0),(7200,0)])

In [19]:
m2.setInputValues("Ti",[(0,273.15+10),(7200,273.15+10)])

In [20]:
m2.setInputValues("Vdi",[(0,4*V/3600),(7200,4*V/3600)])
```
In [21]: m2.setInputValues("xCD21",[(0,400e-6),(7200,400e-6)])

In [22]: m2.setInputValues("N",[(0,15),(2700,15),(2700,5),(3600,5),(3600,15),(6300,15),(6300,5)])

In [23]: m2.simulate()

In [24]: tmn, N = m2.getSolutions(["time","N"])

In [25]: plt.plot(tmn,N,linewidth=1.5)
   plt.title("Number of students in room")
   plt.xlabel(r"time $t$ [min]")
   plt.ylabel(r"$N$")

Out[25]: <matplotlib.text.Text at 0x8986fb0>
In [26]:

```python
import matplotlib.pyplot as plt

plt.plot(time, xCO2, color='green')
plt.title(r'$\text{CO}_2$ concentration in room')
plt.xlabel(r'$\text{time} \ [\text{min}]$')
plt.ylabel(r'$x_{\text{CO}_2} \ [\text{ppm}]$')
```

Out[26]: `<matplotlib.text.Text at 0x87bb470>`

---

In [ ]:
5.2 Test

For each module, test is performed. All the testing that were carried out, were not done in a formal testing methodology, as there are not so many modules/functions. For some of the modules, testing were carried out using standard Python test suit called “unittest”, whereas for other, testing were done just by exploring during the runtime as it was all enough. We tested the getXXXNames(), getXXXValues(), setXXXValues() where XXX = {Continuous, Parameter, Input, Output}, and setXXXOptions() where XXX = {Simulation, Optimization, Linearization}, etc. Tests were also carried out if user enters incorrect information that would generate a meaningful error. We did optimization testing for a Modelica model called "DM.mo", provided by Alachew. We did linearization test and FMU test too. Few testing are presented below:

The following test is performed on the model "FractionCO2Room.mo". The test includes passing incorrect file name, model name, getQuantities(), getContinuousValues() and setInputValues(), etc.
```python
In [1]: from OMPython import ModelicaSystem

In [2]: m2=ModelicaSystem("FractionCO2Ro.mo","FractionCO2Room.ModFractionCO2Room")

2016-03-24 16:05:44,280 - OMCSession - INFO - OMC Server is up and running at file://c:sudeep\appdata\local\temp\openmodelica.objid.6310f14495de45b589cf382c50b9e6c2
INFO:OMCSession:OMC Server is up and running at file://c:sudeep\appdata\local\temp\openmodelica.objid.6310f14495de45b589cf382c50b9e6c2
Error: File does not exist!!!

In [3]: m2=ModelicaSystem("FractionCO2Room.mo","FractionCO2Room.ModFractionCO2m")

2016-03-24 16:06:01,572 - OMCSession - INFO - OMC Server is up and running at file://c:sudeep\appdata\local\temp\openmodelica.objid.22a9a4e2b4e4d4a14b8761767a2e1f
2016-03-24 16:06:01,572 - OMCSession - INFO - OMC Server is up and running at file://c:sudeep\appdata\local\temp\openmodelica.objid.22a9a4e2b4e4d4a14b8761767a2e1f
INFO:OMCSession:OMC Server is up and running at file://c:sudeep\appdata\local\temp\openmodelica.objid.22a9a4e2b4e4d4a14b8761767a2e1f
buildModel Error: Error: Cannot Instantiate FractionCO2Room.ModFractionCO2m due to class specialization packaging.
Error: Class FractionCO2Room.ModFractionCO2m not found in scope <TOP>.

In [4]: m2=ModelicaSystem("FractionCO2Room.mo","FractionCO2Room.ModFractionCO2Room")

2016-03-24 16:06:10,723 - OMCSession - INFO - OMC Server is up and running at file://c:sudeep\appdata\local\temp\openmodelica.objid.7e11934150784dc6a24d2b6edf1ae05
2016-03-24 16:06:10,723 - OMCSession - INFO - OMC Server is up and running at file://c:sudeep\appdata\local\temp\openmodelica.objid.7e11934150784dc6a24d2b6edf1ae05
2016-03-24 16:06:10,723 - OMCSession - INFO - OMC Server is up and running at file://c:sudeep\appdata\local\temp\openmodelica.objid.7e11934150784dc6a24d2b6edf1ae05
INFO:OMCSession:OMC Server is up and running at file://c:sudeep\appdata\local\temp\openmodelica.objid.7e11934150784dc6a24d2b6edf1ae05

In [5]: m2.getQuantities(['ke'])

Error!!! ['ke'] does not exist.

In [6]: m2.getQuantities(['ke', 'Ke', 'V', 'vv'])

Error!!! ['ke', 'vv'] does not exist.

In [7]: cName = m2.getContinuousNames()
```
In [8]: cName
Out[8]: ['U', 'm', 'nCO2', 'der(U)', 'der(m)', 'der(nCO2)', 'H', 'Hde', 'Hdi', 'Hh', 'HHI', 'Qd', 'Qdg', 'Qdi', 'T', 'Tt', 'Vd1', 'mdCO2g', 'mde', 'mdi', 'n', 'ndCO2e', 'ndCO2g', 'ndCO2i', 'nde', 'ndi', 'p', 'pCO2i', 'xCO2', 'xCO2i', 'N']

In [9]: cVal = m2.getContinuousValues(['Tt'])
   
   Error!!! ['Tt'] does not exist.

In [10]: cVal = m2.getContinuousValues('Tt')

   !!! Tt does not exist

In [11]: cVal = m2.getContinuousValues(['T', 'r'])
   
   NameError Traceback (most recent call last)
   <ipython-input-11-d442988671b0> in <module>()
     ----> 1 cVal = m2.getContinuousValues(['T', 'r'])

   NameError: name 'r' is not defined

In [12]: cVal = m2.getContinuousValues(['T', 'pa', 'abc'])
   
   Error!!! ['pa', 'abc'] does not exist.

In [13]:

Enhanced OpenModelica Python Interface
In [14]: cVal = m2.getContinuousValues(['T', 'p'])
Out[14]: [None, None]

In [15]: cVal = m2.getContinuousValues(['T', 'U', 'n'])

In [16]: cVal
Out[16]: [None, None, None]

In [17]: pName = m2.getParameterNames()

In [18]: pName
Out[18]: ['Hhθ', 'Ke', 'MCO2', 'Mair', 'Qdθ0', 'T0', 'U0', 'V', 'etav', 'mθ', 'mdCO2θ', 'nCO2θ', 'pθ', 'pa', 'xCO2θ']

In [19]: pVal = m2.getParameterValues(pName)

In [20]: pVal
Out[20]: [None, 0.000990990099009901, 44.0, 29.0, 100.0, 293.0, None, 135.0, 0.6, None, 1.157407407407407e-05, None, None, None, 1.010000e-05, 0.0004]
In [21]: pVal2 = m2.setParameterValues(['V', 'pa', 'p1'])

---------------------------------------------------------------------------
TypeError: setParameterValues() takes exactly 3 arguments (2 given)

In [22]: pVal = m2.getParameterValues(['V', 'pa', 'p1'])

Error!!! ['p1'] does not exist.

In [23]: pVal

In [24]: pVal = m2.getParameterValues(['V', 'pa'])

In [25]: pVal

Out[25]: [135.0, 101000.0]

In [26]: m2.setParameterValues(['Vv', 'pa'], [137.4, 110000])

Error!!! ['Vv'] does not exist.

In [27]: m2.setParameterValues(['V', 'pa'], ['137.4', '110000'])

Out[27]: 'Error!!! value should not be string'

In [28]: m2.setParameterValues(['V', 'pa', 'm'], [137.4, 110000])

Error!!! ['m'] does not exist.

In [29]: m2.setParameterValues(['V', 'pa', 'm0'], [137.4, 110000])

Error: Both list must be of same length!!!

In [30]: m2.setParameterValues(['V', 'pa'], [137.4, 110000])

In [31]: m2.getQuantities(['V', 'pa'])

Out[31]:

<table>
<thead>
<tr>
<th>Changable</th>
<th>Description</th>
<th>Name</th>
<th>Value</th>
<th>Variability</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Classroom volume, m3</td>
<td>V</td>
<td>137.4</td>
<td>parameter</td>
</tr>
<tr>
<td>true</td>
<td>Atmospheric pressure, Pa</td>
<td>pa</td>
<td>110000.0</td>
<td>parameter</td>
</tr>
</tbody>
</table>

In [32]: inputName = m2.getInputNames()

In [33]: inputName

Out[33]: ['Qdi', 'Ti', 'Vdi', 'xCO2i', 'N']
In [34]: inputVal = m2.getInputValues()

In [35]: inputVal
Out[35]: [None, None, None, None, None]

In [36]: m2.getQuantities(inputName)

Out[36]:

<table>
<thead>
<tr>
<th>Changable</th>
<th>Description</th>
<th>Name</th>
<th>Value</th>
<th>Variability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 false</td>
<td>Heating, kW</td>
<td>Qdi</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>1 false</td>
<td>Influent temperature -- ambient temperature, K</td>
<td>Ti</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>2 false</td>
<td>Influent volumetric flow rate, m3/s</td>
<td>Vdi</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>3 false</td>
<td>Influent mole fraction of CO2, -</td>
<td>xCO2</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>4 false</td>
<td>Number of persons in classroom, -</td>
<td>N</td>
<td>None</td>
<td>continuous</td>
</tr>
</tbody>
</table>

In [37]: m2.setInputValues('N', [0, 1])
Out[37]: 'Error!!! Incorrect format'

In [38]: m2.setInputValues('N', [(0, 1), (0.5, 1), (0.5, 10), (1, 10)])

In [40]: m2.getInputValues('N')
Out[40]: [(0, 1), (0.5, 1), (0.5, 10), (1, 10)]

m2.getQuantities(inputName)

In [41]: m2.getQuantities(inputName) # Need to fix this

Out[41]:

<table>
<thead>
<tr>
<th>Changable</th>
<th>Description</th>
<th>Name</th>
<th>Value</th>
<th>Variability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 false</td>
<td>Heating, kW</td>
<td>Qdi</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>1 false</td>
<td>Influent temperature -- ambient temperature, K</td>
<td>Ti</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>2 false</td>
<td>Influent volumetric flow rate, m3/s</td>
<td>Vdi</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>3 false</td>
<td>Influent mole fraction of CO2, -</td>
<td>xCO2</td>
<td>None</td>
<td>continuous</td>
</tr>
<tr>
<td>4 false</td>
<td>Number of persons in classroom, -</td>
<td>N</td>
<td>None</td>
<td>continuous</td>
</tr>
</tbody>
</table>

In [42]: m2.setInputValues('N', [(0, 1), (0.5, 1), (0.5, 10), (1, 10)])
Out[42]: 'Error!!! Incorrect format'

In [43]: m2.setInputValues('N', [(0, 1), (0.5, 1), (0.5), (1, 10)])
Out[43]: 'Error!!! Incorrect format'

In [ ]:
The optimization and linearization test is also performed on "DM.mo" as below:

```python
In [1]: from OMPython import ModelicaSystem

In [2]: m=ModelicaSystem("DM.mo","diesel_model")

INFO:OMCSession:OMC Server is up and running at file://c:\users\sudeep\appdata\local\temp\openmodelica.objid.7e87c6b8743f46ed2d5a40e9d211b1b

In [3]: m.getOptimizationOptions()

<table>
<thead>
<tr>
<th>Options</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>startTime</td>
</tr>
<tr>
<td>1</td>
<td>stopTime</td>
</tr>
<tr>
<td>2</td>
<td>numberOfIntervals</td>
</tr>
<tr>
<td>3</td>
<td>stepSize</td>
</tr>
<tr>
<td>4</td>
<td>tolerance</td>
</tr>
<tr>
<td>5</td>
<td>simflags</td>
</tr>
</tbody>
</table>

In [4]: m.setOptimizationOptions(stepsize = 0.5, simflags = '-lv LOG_IPOPT -optimizerNP 1')

!!! stepsize is not an option

In [5]: m.setOptimizationOptions(stepSize = 1, simflags = '-lv LOG_IPOPT -optimizerNP 1')

In [6]: m.getOptimizationOptions()

<table>
<thead>
<tr>
<th>Options</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>startTime</td>
</tr>
<tr>
<td>1</td>
<td>stopTime</td>
</tr>
<tr>
<td>2</td>
<td>numberOfIntervals</td>
</tr>
<tr>
<td>3</td>
<td>stepSize</td>
</tr>
<tr>
<td>4</td>
<td>tolerance</td>
</tr>
<tr>
<td>5</td>
<td>simflags</td>
</tr>
</tbody>
</table>

In [7]: m.optimize()

Out[7]:
Enhanced OpenModelica Python Interface
Hessian evaluations = 14\nTotal CPU secs in IPOPT (w/o function evaluations) = 0.086\nTotal CPU secs in NLP function evaluations = 0.007\nEXIT: Optimal Solution Found.

'resultFile': 'C:/Users/sudeep/Desktop/optimize/diesel_model_res.mat',
'simulationOptions': "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 1, tolerance = 1e-008, method = 'optimization', fileNamePrefix = 'diesel_model', options = '', outputFormat = 'mat', variableFilter = '.\*', cflags = '', simflags = '-lv LOG_IPOPT -optimizerNP 1'",
'timeBackend': 0.3609787466361703,
'timeCompile': 8.10525308118312,
'timeFrontend': 0.376367194207981,
'timeSimCode': 0.03515981128644058,
'timeSimulation': 0.2368376902890999,
'timeTemplates': 0.2284692165195957,
'timeTotal': 9.344815784310669

In [8]: m.setOptimizationOptions(stepSize = 0.002, simflags = '-lv LOG_IPOPT -optimizerNP 1')
In [9]: m.getOptimizationOptions()

Out[9]:

<table>
<thead>
<tr>
<th>Options</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 startTime</td>
<td>0.0</td>
</tr>
<tr>
<td>1 stopTime</td>
<td>1.0</td>
</tr>
<tr>
<td>2 numberOfIntervals</td>
<td>500</td>
</tr>
<tr>
<td>3 stepSize</td>
<td>0.002</td>
</tr>
<tr>
<td>4 tolerance</td>
<td>1e-8</td>
</tr>
<tr>
<td>5 simflags</td>
<td>-lv LOG_IPOPT -optimizerNP 1</td>
</tr>
</tbody>
</table>

In [10]: m.optimize()

Number of objective function evaluations = 64
Number of equality constraint evaluations = 81
Number of inequality constraint Jacobian evaluations = 64
Number of equality constraint Jacobian evaluations = 6
Number of Lagrangian Hessian evaluations = 63
Total CPU secs in IPOPT (w/o function evaluations) = 11.345
EXIT: Optimal Solution Found.

'resultFile': 'C:/Users/sudeep/Desktop/optimize/diesel_model_res.mat',
'simulationOptions': "startTime = 0.0, stopTime = 1.0, numberOfIntervals = 50, tolerance = 1e-008, method = 'optimization', fileNamePrefix = 'diesel_model', options = '', outputFormat = 'mat', variableFilter = '.\*', cflags = '', simflags = '-lv LOG_IPOPT -optimizerNP 1'",
'timeBackend': 0.3718442803743409,
'timeCompile': 6.949053745876261,
'timeFrontend': 0.5024228258419493,
'timeSimCode': 0.02108717800655724,
'timeSimulation': 14.93316356117385,
'timeTemplates': 0.2440582046653263,
'timeTotal': 23.0506679489261
The `m.linearize()` generates a linearized Modelica model "linear_diesel_model.mo" with matrices $A$, $B$, $C$ and $D$ as shown in table 5-2.

Table 5-2 Resulted linear_diesel_model.mo

```
model linear_diesel__model
  parameter Integer n = 4; // states
  parameter Integer k = 2; // top-level inputs
  parameter Integer l = 4; // top-level outputs
  parameter Real x0[4] =
    [0.3810129061256339,0.5410314408357427,0.5130706277970849,0.2727549686665666];
  parameter Real u0[2] = [1.433321695179709e-005,0.003951604388916958];
  parameter Real A[4,4] = [-22.68841795052424,0.080912017810522,3.796808876909464;0.01325511204778,5.96599597069883,0.4085544056220567,-0.281478532273065];
  parameter Real B[4,2] = [-22.68841795052424,0.080912017810522,3.796808876909464;0.01325511204778,5.96599597069883,0.4085544056220567,-0.281478532273065];
  parameter Real C[4,4] = [-22.68841795052424,0.080912017810522,3.796808876909464;0.01325511204778,5.96599597069883,0.4085544056220567,-0.281478532273065];
```

Enhanced OpenModelica Python Interface
parameter Real D[4,2] = [0,0;0,0;0,0;0,0];
Real x[4](start=x0);
input Real u[2](start= u0);
output Real y[4];

Real x_Pp_em = x[1];
Real x_Pp_im = x[2];
Real x_Pw_ice = x[3];
Real x_Pw_tc = x[4];
    Real u_Pu_f = u[1];
    Real u_Pu_wg = u[2];
    Real y_P$OMC$objectLagrangeTerm = y[1];
    Real y_P$OMC$objectMayerTerm = y[2];
    Real y_Pcost1 = y[3];
    Real y_Pcost2 = y[4];

equation
    der(x) = A * x + B * u;
    y = C * x + D * u;
end linear_diesel__model;
Testing for linearizing and converting Modelica model "BouncingBall.mot" into FMU.

<table>
<thead>
<tr>
<th>In [1]:</th>
<th>from OMPython import ModelicaSystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>In [2]:</td>
<td>m=ModelicaSystem(&quot;BouncingBall.mo&quot;,&quot;BouncingBall&quot;)</td>
</tr>
<tr>
<td>2016-03-24 19:08:52,463 - OMCSession - INFO - OMC Server is up and running at file://c:/users/sudeep/appdata\local\temp\openmodelica.objid.2125421c21e64d389bb6347b319b5bc4</td>
<td></td>
</tr>
<tr>
<td>INFO:OMCSession:OMC Server is up and running at file://c:/users/sudeep/appdata\local\temp\openmodelica.objid.2125421c21e64d389bb6347b319b5bc4</td>
<td></td>
</tr>
<tr>
<td>In [3]:</td>
<td>m.linearize()</td>
</tr>
</tbody>
</table>
| Out[3]: | {'messages': 'stdout | info | Linearization will performed at point of time: 1.000000\nstdout | info | Linear model is created!\n', 'resultFile': 'C:/Users/sudeep/Desktop/model/bouncingball/BouncingBall_res.mat', 'simulationOptions': 'startTime = 0.0, stopTime = 1.0, numberOfIntervals = 500, tolerance = 1e-008, method = 'dassl', fileNamePrefix = 'BouncingBall', options = '', outputFormat = 'mat', variableFilter = '.+', cflags = ' ', simflags = ' ', 'timeBackend': 0.0498136946825125, 'timeCompile': 6.80080095057673, 'timeFrontend': 0.0366141783570892, 'timeSimCode': 0.02022067749118802, 'timeSimulation': 0.3706763661283618, 'timeTemplates': 0.5123245351036378, 'timeTotal': 7.33054647503689}
| In [4]: | m.convertMo2Fmu() |
| Out[4]: | 'SimCode: The model BouncingBall has been translated to FMU' |
| In []: |

The `m.linearize()` generates a linearized Modelica model "linear_BouncingBall.mo" as shown in table 5-3. The `m.convertMo2Fmu()` generates a FMU file called "BouncingBall.fmu".

**Table 5-3 Resulted linear_BouncingBall.mo**

<table>
<thead>
<tr>
<th>model linear_BouncingBall</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter Integer n = 2; // states</td>
</tr>
<tr>
<td>parameter Integer k = 0; // top-level inputs</td>
</tr>
<tr>
<td>parameter Integer l = 0; // top-level outputs</td>
</tr>
<tr>
<td>parameter Real x[2] = [0.94412639080022,0.8391126390914951];</td>
</tr>
<tr>
<td>parameter Real u[0] = [i for i in 1:0];</td>
</tr>
<tr>
<td>parameter Real A[2,2] = [0,1;0,0];</td>
</tr>
<tr>
<td>parameter Real B[2,0] = zeros(2,0);</td>
</tr>
</tbody>
</table>
parameter Real C[0,2] = zeros(0,2);
parameter Real D[0,0] = zeros(0,0);
Real x[2](start=x0);
input Real u[0];
output Real y[0];

Real x_Ph = x[1];
Real x_Pv = x[2];

equation
  der(x) = A * x + B * u;
  y = C * x + D * u;
end linear_BouncingBall;
6 Issues and Limitation

6.1 Regarding System

- Supports only in Python 32-bit
- If Python 2.7 is used, then CORBA, has to be installed separately. The details about the installation can be found in the paper "Design and Implementation of a User Friendly OpenModelica - Python Interface" by Anand Kalaiarasi Ganeson
- May not support if Anti-Virus program is running

6.2 Regarding Application

- Does not support plot() API of OpenModelica
- Regarding the linearized model, currently it only creates linearized model that contains A, B, C and D matrices but does not provide an access/retrieve those matrices using Python object.
7 Future Work

As a future work, one can add more features that provide users with more flexibility such as:

- Change setInputValues() to more efficient and flexible form
- Create a method that could retrieve matrices A, B, C and D of linearized model
- Currently, simulation, optimization and linearization options are only provided with few major options such as; startTime, stopTime, stemSize, solver, etc. But there are more options. So those can be added too in future too.
- Optimization and linearization are not complete yet, so need improvement in optimization and linearization
8 Conclusions

This project is completely focused on OpenModelica using enhanced Python interface that provides an environment for users to access the OMC's result via Python object rather than dealing with modeling Modelica models or adding extra features to OpenModelica. This project is developed in order to replace the strong computational tool, MATLAB, though it might not be at same level but to some useful level. Thus, providing users with a power to do further data manipulation of the result obtained from the OMC. Hence, it allows users to simulate model, values can be get/set for quantities and do re-simulation with the corresponding changed values. Additionally, it supports conversion from Modelica model to FMU and vice-versa, optimizing and linearizing model using a Python object.
9 Appendix

9.1 Tools used

- Windows 8
- Notepad ++
  - Editor for coding. Free source code editor, fast and easy to write code.
- OpenModelica 1.9.4-dev-beta2 (OpenMoelica 1.9.3, OM1.9.4-dev-beta1 previously)
  - Nightly build released (2016/01/08)
  - Downloaded from "https://build.openmodelica.org/omc/builds/windows/nightly-builds/older/"
- Python 2.7.11 (32-bit)
- Command Prompt
  - Shell to run and check result. Fast and easy to use.
- Jupyter Notebook
  - It is a free, open source web application that allows running code, equations and explanatory text. Used for demonstration.

9.2 Installation Procedure

- Install Python 2.7 (32-bit though the Operating System is 64-bit)
  - Download from "https://www.python.org/downloads/release/python-2710/
- Install module pyparsing.py
  - Download from "https://pyparsing.wikispaces.com/Download+and+Installation"
- Install OpenModelica1.9.4-dev.beta2 (for now replace OMPython module with this version of OMPython)
- Install "setup.py" which is located at "C:\OpenModelica1.9.4-dev.beta2\share\omc\scripts\PythonInterface"
  - Run the command in command prompt or any Python shell as "python setup.py install"
10 References

List of References:

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns detlösningar av teknisk och administrativ art.

Uphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Sudeep Bajracharya