Python Framework – For structural dynamics

User´s Guide

Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik
Fachgebiet Softwaretechnik

ABSTRACT

This document represents the user guide that illustrates how to install and run the Python Framework for modeling and simualtion of variable-structure models under Windows OS. The aim is to enable the user to create his own models and simulate them with the given Framework. An introduction to the Framework is given and examples explain how to use the Framework.

Contents

1 Introduction

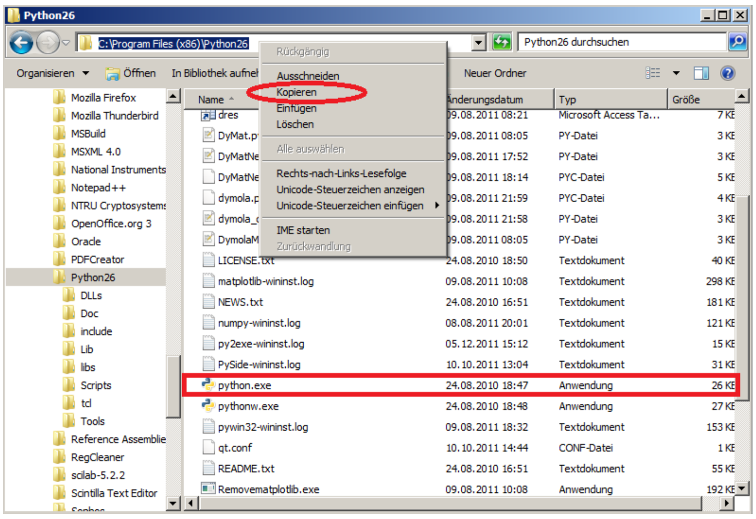
In this section we show you how to set up DySMo. If you have any questions or remarks please write an email to a.mehlhase@tu-berlin.de or leave a comment at `https://bitbucket.org/amehlhase/dysmo`.

1.1 Before you start

Save the install folder 'DySMo' to your desired destination path. The directory references are relative paths for the rest of the document. If there is a need to change some configurations «yourDirectory» is used.

1.2 Install Python

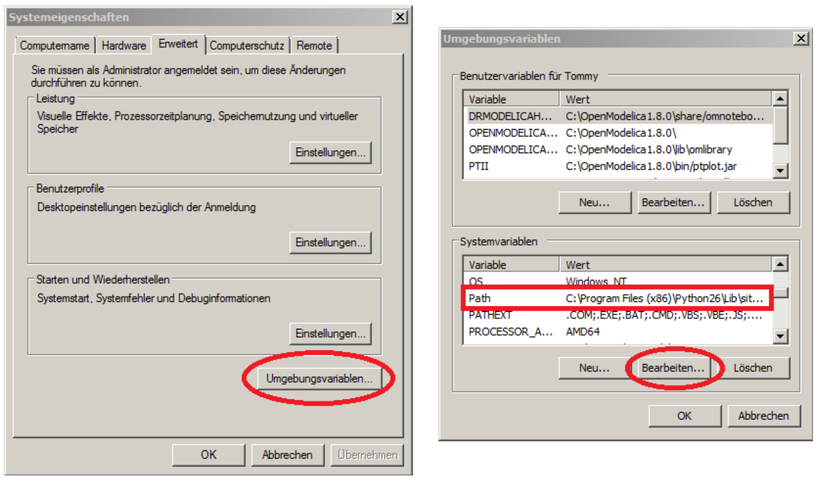
To install the *python-interpreter* under WindowsOS you can use the link in the software.txt file. It is strongly recommended to use the **Version 2.6**, the source code is not tested with other version. Therefore there is no guarantee that everything will work as expected. For further information about the python-interpreter, refer to the official Python project page `www.python.org`. Please also install all necessary packages from the software.txt. Please use the recommended versions, otherwise the framework might not work.



Name	Änderungsdatum	Typ	Größe
dres	09.08.2011 08:21	Microsoft Access Ta...	7 KE
DyMat.p	09.08.2011 08:05	PY-Datei	3 KE
DyMatNe	09.08.2011 17:52	PY-Datei	3 KE
DyMatNe	09.08.2011 18:14	PYC-Datei	5 KE
dymola.p	09.08.2011 21:59	PYC-Datei	4 KE
dymola_	09.08.2011 21:58	PY-Datei	3 KE
DymolaM	09.08.2011 08:05	PY-Datei	3 KE
LICENSE.txt	24.08.2010 18:50	Textdokument	40 KE
matplotlib-wininst.log	09.08.2011 10:08	Textdokument	298 KE
NEWS.txt	24.08.2010 16:51	Textdokument	181 KE
numpy-wininst.log	08.08.2011 20:01	Textdokument	121 KE
py2exe-wininst.log	05.12.2011 15:12	Textdokument	15 KE
PySide-wininst.log	10.10.2011 13:04	Textdokument	31 KE
python.exe	24.08.2010 18:47	Anwendung	26 KE
pythonw.exe	24.08.2010 18:48	Anwendung	27 KE
pywin32-wininst.log	09.08.2011 18:32	Textdokument	153 KE
qt.conf	10.10.2011 14:44	CONF-Datei	1 KE
README.txt	24.08.2010 16:51	Textdokument	55 KE
Removematplotlib.exe	09.08.2011 10:08	Anwendung	192 KE

Figure 1: Copy this **path** and paste it to your environment Variables

After the installation you have to set an environment variable, so that it is possible to run the python interpreter from any path. Therefore locate the file *python.exe*. If Python is installed in the default directory, you can find the file under $C:\backslash ProgramFiles(x86)\backslash Python26$, refer to figure 1. Copy this path and open the *environment variables setup (Systemcontrol->System->advanced->environment variables) (Systemsteuerung->System->Erweiterte Systemeigenschaften->Umgebungsvariablen)*, refer to figure 2. Find the variable **Path** and add the copied path from figure 1 separated by a semicolon. If everything works, you can now open the command line (Eingabeaufforderung) and start the python interpreter. To test it, goto *Start->programs->accessories->execute (Start->Programme->Zubehör->Ausführen)* and type in **cmd**. You can now type in **python**, the interpreter should



Systemeigenschaften

Computername | Hardware | Erweitert | Computerschutz | Remote

Sie müssen als Administrator angemeldet sein, um diese Änderungen durchführen zu können.

Leistung
Visuelle Effekte, Prozessorzeitplanung, Speichernutzung und virtueller Speicher

Benutzerprofile
Desktopeinstellungen bezüglich der Anmeldung

Starten und Wiederherstellen
Systemstart, Systemfehler und Debuginformationen

Umgebungsvariablen

Benutzervariablen für Tommy

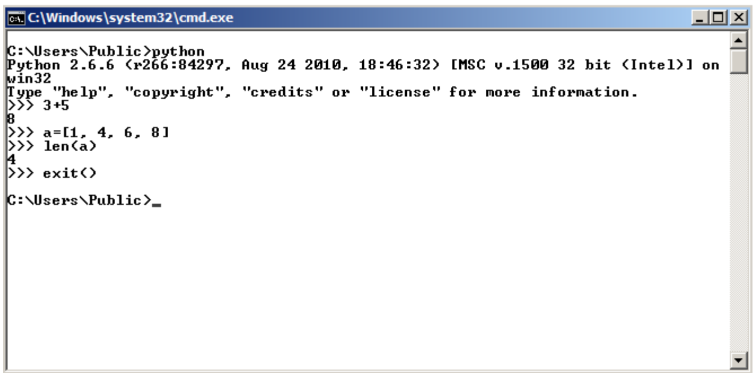
Variable	Wert
DRMODELICAH...	C:\OpenModelica1.8.0\share/omnotebo...
OPENMODELICA...	C:\OpenModelica1.8.0\
OPENMODELICA...	C:\OpenModelica1.8.0\lib\omlibrary
PTII	C:\OpenModelica1.8.0\bin/ptplot.jar

Systemvariablen

Variable	Wert
OS	Windows_NT
Path	C:\Program Files (x86)\Python26\Lib\sit...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	AMD64

Figure 2: Edit the environment variable **Path**

start and accept your commands, figure 3.



```
C:\Users\Public>python
Python 2.6.6 (r266:84297, Aug 24 2010, 18:46:32) [MSC v.1500 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 3+5
8
>>> a=[1, 4, 6, 8]
>>> len(a)
4
>>> exit()

C:\Users\Public>_
```

Figure 3: Open your console and test some python commands

1.3 Install Dymola

To install the modeling environment Dymola, please refer to the Dymola user´s manual («your-Directory»\ Documentation\ Dymola5Manual.pdf). Make sure you have a fully licensed version, with the demo version the following examples do not work. Furthermore the examples do mostly not work with a borrowed server license (it is supposed to work with the new Version Dymola 2013, but was not tested yet)). For more information about Dymola visit `www.dymola.com`.

1.4 Install OpenModelica

You can also use OpenModelica to simulate your models. DySMo works with OpenModelica Version 1.8. A link to the installation for OpenModelica on Windows is also given in the software.txt. For further information about OpenModelica, refer to the official project page `www.openmodelica.org`.

1.5 Install Matlab/Simulink

DySMo does also support Matlab/Simulink. To use Matlab/Simulink in the Framework Matlab/Simulink has to be installed on your computer. The Framework was tested with Matlab 2011a and was not fully tested with other versions, so the Framework might not work with other versions.

2 Getting started with DySMo

DySMo supports different modeling tools and an arbitrary number of mode switches. This chapter gives you a design overview and shows you how you have to prepare your models so that they can be used as variable-structure models in the given Framework. In the second part the focus is on how to run the delivered examples.

2.1 Design

The Framework is implemented through different objects which describe the variable-structure model. Figure 4 presents an overview. The main files are located in «yourDirectory» \ dysmo \ source. For every new variable-structure-model you have to create a parameter file, <model-Name.py>. The body of such a file is shown in chapter 2.3.

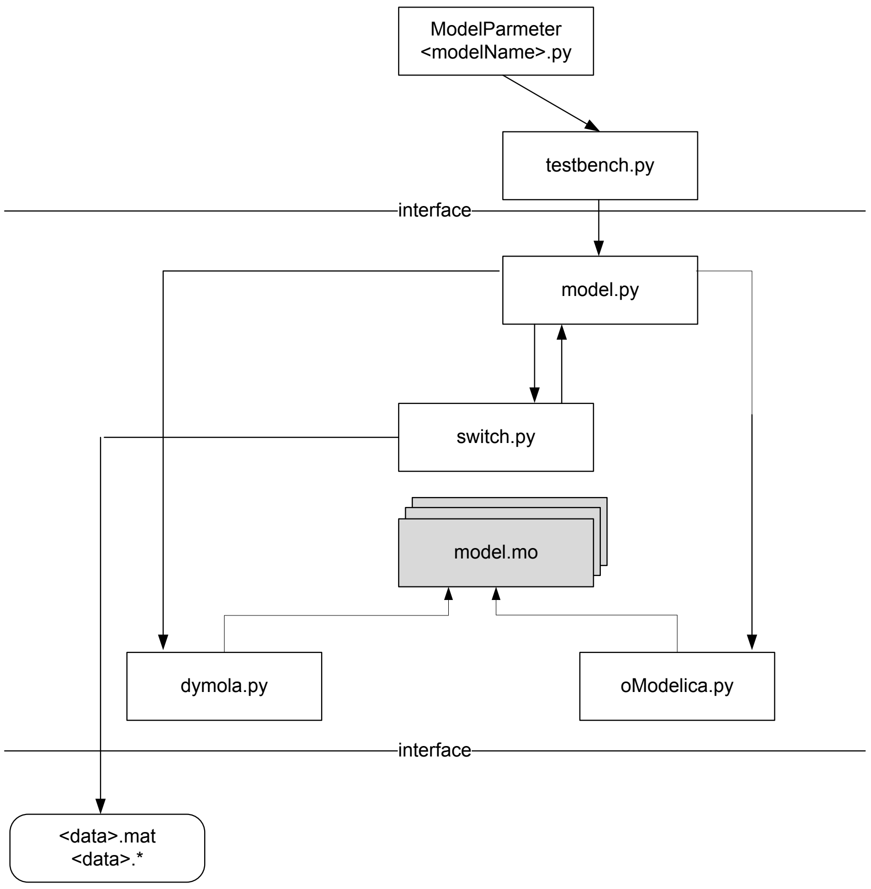


Figure 4: Static framework overview

This parameter file is given to the *main.py* which is the executable Python file which needs to be started. The actual variable-structure model inside the Python Framework is then build by the *main.py*. After building the variable-structure model the model is handed to the method *switch.py*. This method handles the actual mode switch, the initialization of the new mode and the saving of the simulation results of each simulation. In the following a few examples are introduced to show how the Framework can be used.

2.2 Case study: The extended bouncing ball

In this section a case study is designed to illustrate a simulation cycle using the Framework. You can find the same model implemented under «yourDirectory»\sample \ mechStruk.

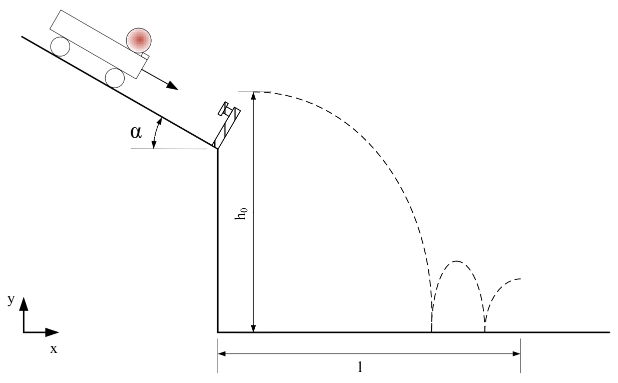


Figure 5: Physical representation: The extended bouncing ball

For the given model, three modes are needed (wagen_struc ID=1, contact_struc ID=2, ball_struc ID=3). Every mode has a terminate condition, for example the model terminates from wagon to ball when the end of the ramp is reached (mode 1->2). These conditions have to be specified in the corresponding model (refer to «yourDirectory»\sample \ mechStruk \ mechanik.mo). When this condition is reached a variable called switch_to (this variable is expected to have this name) has to be set, to the number of the mode to switch to. Figure 5 shows how the system to the model might look like.
A sequence diagram which shows the sequence of the simulation is shown in figure 6. (You can switch translation/compilation of the model on / off in *main.py*, refer to chapter 2.4).

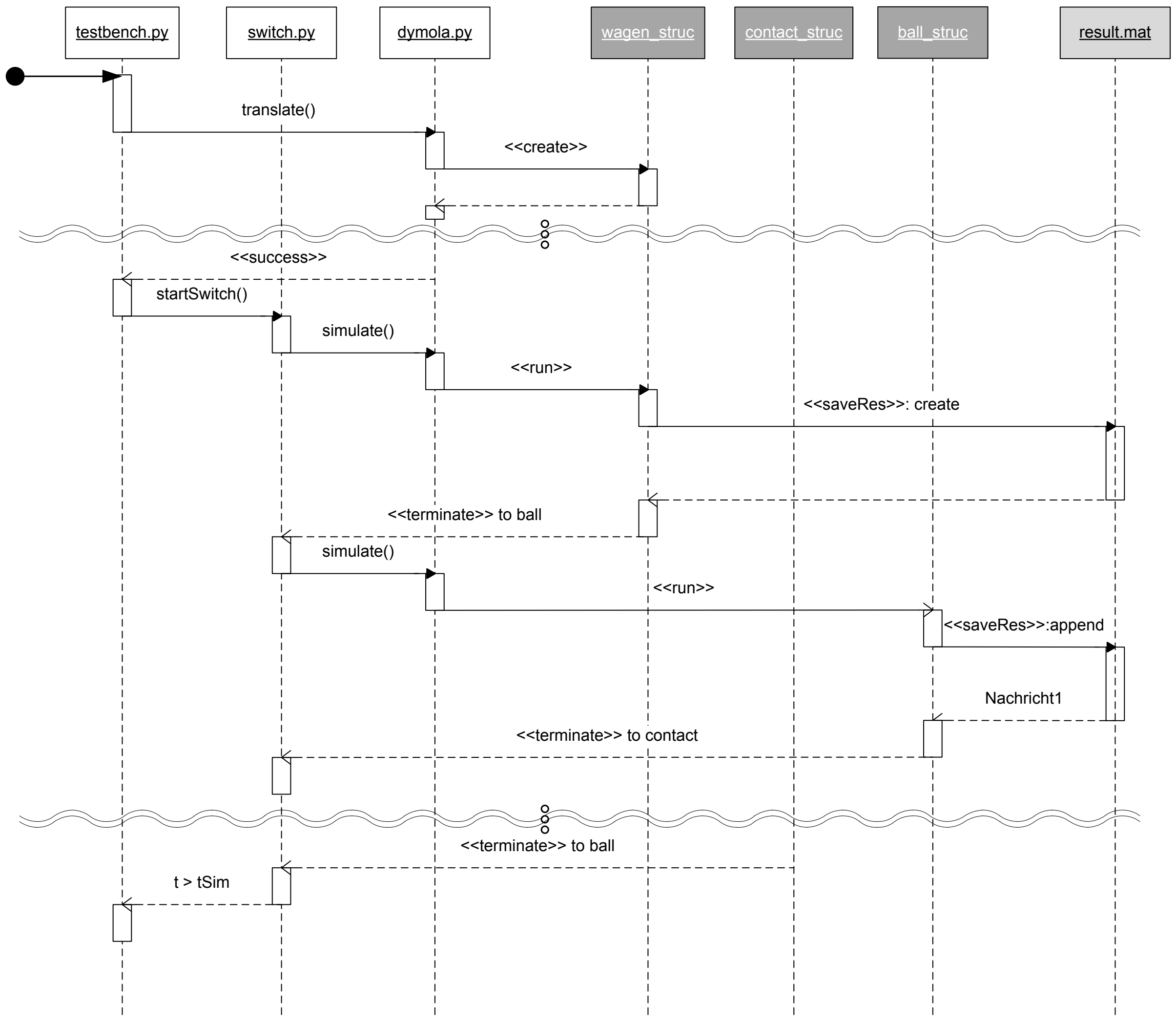


Figure 6: Sequence Diagram: The extended bouncing ball

TODO

Figure 7: Figure of objects

2.3 Body of the parameter file

Here the model parameter file is described. You can find the complete file in your folder *sample\mechStruk*. The idea is to specify an object-oriented structure of you model. This models are quite easy to specify. The object structure can be seen in figure **??**.

First you have to set your simulation settings in the object simInfo. Parameters like Interval length, Number of Intervals, Tolerance, fixed stepsize and simulation time have to be set here, refer to listing 1. If you need further information for these parameters, refer to the Modelica users guide, the necessary information is based on Modelica models. In Simulink only the stopTime is used as valuable information. Otherwise the intern model settings are used (this will change in future versions).

```
simInfo.startTime = 0           # SET: start time of simulation
simInfo.stopTime = 10           # SET: simulation time
simInfo.solver = Solver.DASSL # SET: default solver for the model
simInfo.tolerance = 3e-05       # tolerance of solver
simInfo.intervalNum = 500       # number of saved data
simInfo.intervalLen = 0         # interval length for saved data
simInfo.fixed = 0               # fixed step size
```

Listing 1: Simulation parameters

Follwing the global settings of the variable-structure model needs to be set. First an instance of a model is created (do not change this). Then the simInfo, that was already defined, is added to the model (don't change that). Next changes are necessary. There the Model files need to be specified. If more than one file is needed please separate them with a comma. The directory in which the models are lying is specified in model.modelPath. Here only one path is expected, for future versions it might be possible to give more than one directory. in model.resFolder a result folder can be specified this folder will be created in the modelPath and is used to save the simulation data. In model.arrToSave the names for the variables to be saved after the simulation can be specified. These names will be the names of the names of the saved columns. In the plotList the data to be plotted after the simulation through Python can be specified. You always have to specify a variable for he x-axis and the y-axis. The variable "‘t"’ is always the time and saved as default.

```
model = d.model() ### DO NOT CHANGE###

model.simInfo = simInfo ### DO NOT CHANGE###

model.moFile = ["mechanik.mo"]
model.modelPath = "..\..\sample\MechStruk"
model.resFolder = "result"
model.arrToSave = ['x', 'y']
model.plotList = [['t', 'y']]
```

Listing 2: Path settings

```
model1 = d.mode() # instance of a new mode

model1.modeName = "mechanik.wagen_struc" # SET: name of the model for this mode
model1.tool = Env.DYMOLA # SET: tool which is used for the simulation of this mode
model1.arrToSave = ['x', 'y'] # SET: variables to be observed, these will be saved
    under the specified names in the model
model1.simInfo = d.simInfo() # SET: empty solver settings, means the global settings
    are used
#### Example on how to change specific solver information just for this mode
#model1.simInfo.solver = Solver.RADAU
#model1.simInfo.tolerance = 1e-03
```

Listing 3: Specifying a mode

In listing 3 you see how a new mode can be specified. Fist a new instance is created. Then the name of the model which represents the mode is set. Here the first mode is called "‘mechanik.wagen_struc"’. The attribute "‘tool"’ shows which tool should be used to simulate the mode. The modeler can choose between Env.DYMOLA, Env.OMODELICA, and Env. SIMULINK. The user than has to (this will be optional in later versions) specify the variables that should be observed. This variables will be mapped to the array which was set in model.arrToSave. Each mode has simulation informations saved in simInfo. If this object is empty, as is default, the global simulation informations form mode.simInfo will be used. The data can be overwritten if wanted.
In the next step you have to specify the transitions leading out of this mode. In listing 4 a new transition is instantiated. Then the ID to where this transition leads is set. Afterwards the names of the variables which should be read from the current mode are set. These values are then mapped to the names of the variables from the new mode with the new mode ID.

```
trans1_2 = d.trans() # creates ne instance of a transistion

trans1_2.modeIDToSw = 2 # SET: mode ID of the next mode
trans1_2.outName = ['x', 'y', 'der(x)', 'der(y)'] # SET: variable names to read
    from model1
trans1_2.inName = ['x', 'h', 'vx', 'vy'] # SET: corresponding variable names to be
    initialized in the new mode
model1.transitions = [trans1_2] # transition is added to the mode
```

Listing 4: Map Variables and create transistions

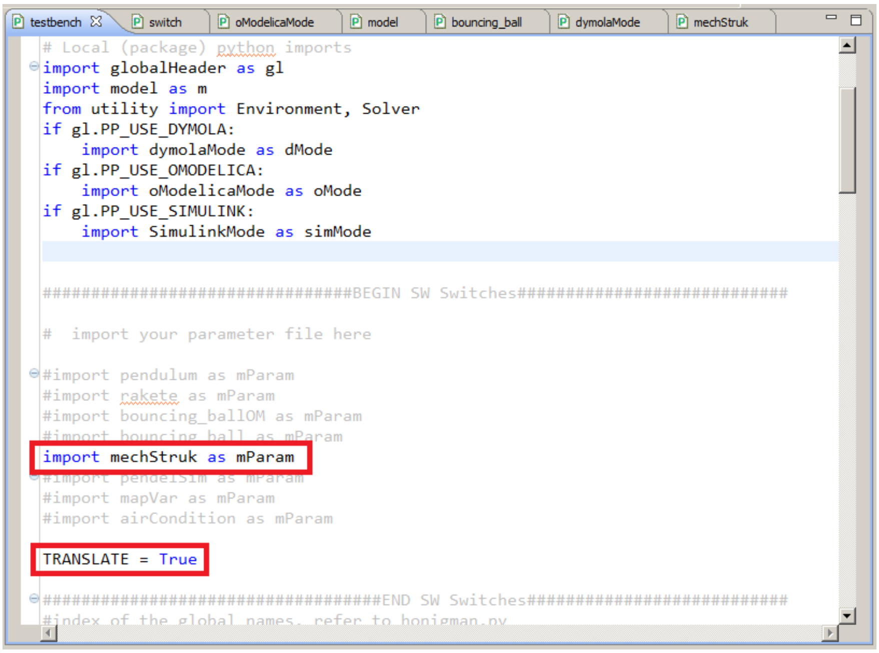
This has to be repeated for each desired mode.
As a last step all defined modes have to be added to the model 5.

```
model.modes = [model1, mode2, mode3]
```

Listing 5: Add modes to model

2.4 Simulate a given example

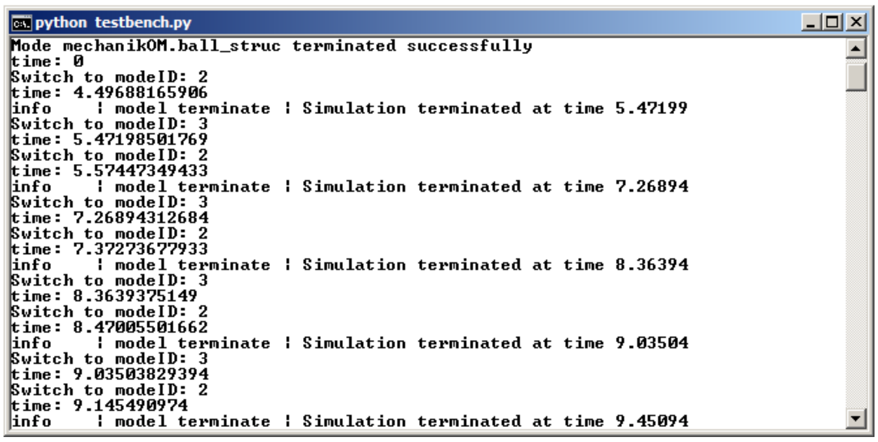
Now that all preconditions are fulfilled, we can devote to the simulation. Note there are more possibilities to start the scripting via the interface, refer to figure 4. This subsection shows how to start it via the command window.
Open the file *main.py* and uncomment the parameter file mechStruk, figure 8. Make sure that all other entries are commented. You can uncomment them later to get familiar with the framework. Set the *TRANSLATE*-flag to *True*, this ensures that your model will be compiled and all necessary folders will be created.



```
# Local (package) python imports
import globalHeader as gl
import model as m
from utility import Environment, Solver
if gl.PP_USE_DYMOLA:
    import dymolaMode as dMode
if gl.PP_USE_OMODELICA:
    import oModelicaMode as oMode
if gl.PP_USE_SIMULINK:
    import SimulinkMode as simMode


################################BEGIN SW Switches############################

#  import your parameter file here

#import pendulum as mParam
#import rakete as mParam
#import bouncing_ballOM as mParam
#import bouncing_ball as mParam
import mechStruk as mParam
#import pendelSim as mParam
#import mapVar as mParam
#import airCondition as mParam

TRANSLATE = True

###################################END SW Switches###########################
#index of the global names, refer to honigman.py
```

Figure 8: Settings in main.py

Now open the console, (as shown in figure 3) and start the scripting with **python main.py**. If everything is ok, Dymola starts and translate your model. Also OpenModelica translates a mode in the background. After few seconds you should be able to pursuit the simulation process, as shown in figure 9.
Another few seconds later a plot with the results should open, like in figure 10. The different colors indicate the different modes.
Now you can try to change the simulation parameters, maybe plot *x over y*. Now you are also able to simulate the other delivered examples.



```
Mode mechanikOM.ball_struc terminated successfully
time: 0
Switch to modeID: 2
time: 4.49688165906
info     | model terminate | Simulation terminated at time 5.47199
Switch to modeID: 3
time: 5.47198501769
Switch to modeID: 2
time: 5.57447349433
info     | model terminate | Simulation terminated at time 7.26894
Switch to modeID: 3
time: 7.26894312684
Switch to modeID: 2
time: 7.37273677933
info     | model terminate | Simulation terminated at time 8.36394
Switch to modeID: 3
time: 8.3639375149
Switch to modeID: 2
time: 8.47005501662
info     | model terminate | Simulation terminated at time 9.03504
Switch to modeID: 3
time: 9.03503829394
Switch to modeID: 2
time: 9.145490974
info     | model terminate | Simulation terminated at time 9.45094
```

Figure 9: Information during the simulation process

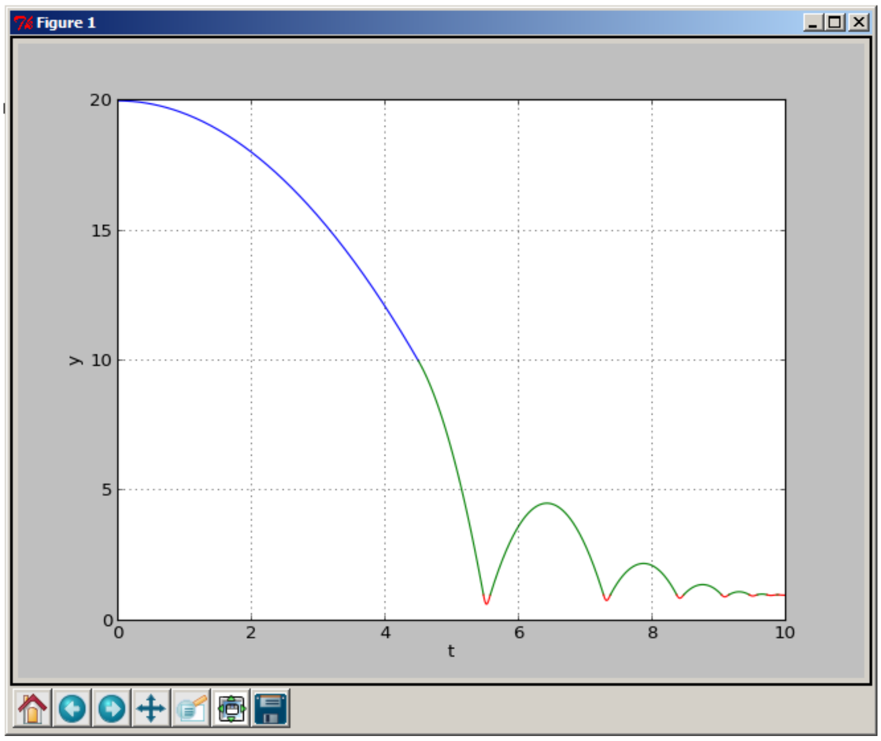


Figure 10: Graphical representation of the simulation results

References

List of Figures