A Python Package for Simulating Variable-Structure Models with Dymola

Alexandra Mehlhase

Department of Software Engineering and Theoretical Computer Science, TU Berlin, Ernst-Reuter-Platz 7, 10587 Berlin, Germany; *a.mehlhase@tu-berlin.de*

SNE Simulation Notes Europe SNE 23(1), 2013, 17-24
DOI: 10.11128/sne.23.tn.10165
Received: Jan. 20, 2013; (Selected MATHMOD 2012 Postconf. Publ.); Revised Accepted: March 20, 2013;

Abstract. It becomes increasingly important to create more accurate models that can be simulated fast. To accomplish this we need models which can change their set of equations during runtime. These models are called variable-structure models. These models enable a user to specify a model with more than one mode and change between these modes during runtime. This can make a simulation faster and in some cases even more accurate. In this paper we present a Python package that enables the user to specify such models in an easy and intuitive manner. The introduced package provides means to use existing Dymola models as modes and simulate the variable-structure model with the Dymola simulation engine. Different examples are presented which were simulated with the new package and the advantages of variable-structure modeling with the Python package is discussed. Furthermore, requirements a model needs to fulfill to be used in a variable-structure model are explained.

Introduction

To study the behavior of a technical system early in the design phase (simulation-) models are often used. Such a model consists of variables and equations which specify the behavior of the model over time. The models are usually described through differential-algebraic equations (DAE). The models are then simulated with a numerical solver. The results of such a simulation can be used to analyze the behavior of a technical system without having to build the real system. Through the ever growing complexity of the real technical systems, the complexity of the models also needs to rise. This leads to the problem that the simulation of a model might become too slow or that some systems cannot be modeled at all. We regard variable-structure models as a solution for this problem. These models consist of different modes between which they can switch. This means that a model can run through different 'modes', where each mode effectively is a model with its own set of equations which describe the physical behavior of this particular mode. When switching from one mode to the next one, the new mode needs to be initialized through the end values of the old mode.

For instance variable-structure models enable the user to model systems that change their behavior. An example for such a model is an airplane which is first a rolling vehicle then a cross between a rolling vehicle and a flying object and then becomes a flying object. In this paper we call models that change their equations in order to model different behavior 'variable-behavior models'.

Another example for variable-structure models is a model which changes its level of detail. With such a model the simulation can be as detailed as necessary and as easy as possible throughout the simulation. For instance if a model can reach critical regions and in these a complex model is needed but otherwise an easier model is sufficient, it would be feasible to use the less detailed model and only switch to the more complex model if needed. We call such models 'variable-detail models' and will show in the evaluation section that such models can save simulation time without significant accuracy losses. Of course there are models that are hard to place in only one category but usually the goals for these two are different. For variable-detail models the goal is to save simulation time or enhance the accuracy through the mode switch. For the variable-behavior models the simulation time is not the main goal but that a system can be simulated at all through the variable-structure approach.

A variable-structure model has always exactly one current mode and switches from mode to mode through defined transition. These transitions hold the information on how the simulation data of the old mode should be used to initialize the new model.

Common simulation tools like Simulink from [1] and Dymola from [2] do not support the change of the variables and the set of equations during simulation. There do exist simulation environments which enable a user to simulate variable-structure models. A brief overview of these environments and approaches are given here.

MOSILAB is a simulation environment based on Modelica which can be used to model and simulate variable-structure models. This tool enhances the Modelica language and uses a Statecharts view to specify the mode switches, see [3]. There is no index reduction implemented in the tool and thus only index-0 models are allowed.

The language SOL was developed by [4] and is an experimental Modelica like language. This language supports modeling variable-structure models, interpreting them and simulating them. An advantage of SOL is that when a mode switch occurs and the causalisation of the model changes only the necessary causalisations are done. This makes the mode switch quite elegant. SOL is an experimental language and thus far not available for large models, hopefully the results will sometime lead to a modeling language which supports variable-structure models. For now a user cannot work with SOL and reuse models from other tools all models would have to be specified in the new language. Another possibility for variable-structure models is Hydra which is described in [5]. Hydra is a language under development which supports variable-structure modeling. It is based on functional programming languages and is therefore not as easy to learn for modelers.

All these possibilities have great ideas and do different things better than our approach but to model and simulation variable-structure models the user has to learn a new language and remodel existing models. With the approach we present a common modeling tool can be used and existing models can be reused. Python as a free language is used for the package, so the package is accessible to anyone interested.

[6] describe an algorithm which transforms a variable-structure model to a normal model by reformulating the modes into one mode. This approach does work but makes the resulting model quite large and will therefore extend the simulation time. This does not seem to be feasible for variable-detail models because simulation time will most likely not be saved. In Simulink enable blocks can be used to model different modes, but the definition of the transitions becomes rather complicated and the blocks that are disabled still take up simulation time, see [7] for more information. In Dymola a mode switch is possible, as long as the variables do not change and the causality of the equations does not change. If either needs to change, both equation systems need to be implemented in the model and through if-statements the switching needs to be defined. [8] describe a possibility for variable-structure models with if-statements in Modelica. Here the equations are reformulated, so depending on the mode the model is in, a multiplication with zero or one takes place. The equations therefore change during simulation. This approach does work but for large models with many mode switches it will become complicated. The equation system is also rather large and might slow down the simulation compared to a real mode switch.
In this paper we will present an easy to use approach to define variable-structure models in Python and use the simulation tool Dymola for the simulation. This package enables the user to reuse existing Dymola models and still work with variable-structure models .

Section 2 introduces the Python package with its design and usability. In Section 3 different examples of variable-structure models which were modeled with the new Python package are presented. The last section provides the conclusion and future work.

1 A New Python Package

This section first gives an overview of how a variable-structure model can be modeled with its different modes and transitions. The design of the package and how the package can be used is afterwards explained.

1.1 A variable-structure model

As was already said in the introduction a variable-structure model is a model which consists of an arbitrary number of modes between which the model can switch. A model can switch from one mode to another mode via one transition. To model such a behavior an object-oriented approach seems feasible. Figure 1 shows how a variable-structure model can be modeled through objects. The ModelObject consists of different modes and each mode can have transitions.

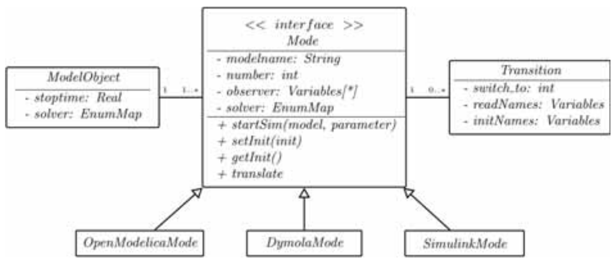

Figure 1: Class structure of a variable-structure model for different modeling environments.

This ModelObject holds all the necessary information about the variable-structure model. It defines the global stop time of the model, a default solver and the modes of the variable-structure model . To be able to integrate different simulation environments into the package an interface which is tool independent was defined. The most important attributes of a mode are:

- a unique mode number to identify the mode
- a model name with path to the original model
- observer variables which will be stored in a data matrix
- a specific solver which overwrites the default solver

Necessary methods in a mode class:

- start simulation (startSim) which starts the simulation in a specific simulation environment
- set initial values (setInit) which sets the initial values in the tool specific init file
- read end values (readEnd) which reads the necessary end values of the tool specific result data file
- translate model (translate) which compiles the model, if necessary

When a specific simulation environment needs to be added new class which implements the mode-interfaces has to be created. The other parts of the model do not have to be changes, as long as the given interface is not changed. For now Dymola is implemented and integrating OpenModelica and Simulink is planned. Each mode can have an arbitrary number of transitions which lead to the next modes. A transition is another class which defines the mode switch and is independent of the used simulation environment. In each transition an attribute exists which holds the identification number of the next mode. Furthermore the information on how the data of the old mode is used to initialize the new mode is stored in the transition.

1.2 Design of the package

In the previous section the object-oriented design of a variable-structure model was explained. This design is used in the Python package which makes it possible to integrate different simulation environments. For a modeler who is used to modeling in simulation environments it might be difficult to define this modeling structure in Python. Therefore the package provides a template to specify the variable-structure model. The user does not need any programming knowledge to be able to use this template.

The package uses the user given information to generate the necessary ModelObject. The basic idea of our approach is to use common simulation environment to simulate a variable-structure model and therefore use their capabilities. It is not the idea to create a new language as was done by [4] with SOL or by [5] with Hydra or with the tool [9, 10]. Our idea is to create a new modeling layer which can manage different modeling environments and which handles the switch from one mode to another during a simulation run. To accomplish this each mode of a variable-structure model needs to be an independent model which consists of variables and equations. Each of these modes has a stop condition which stops the simulation of this particular mode and defines the next mode. Figure 2 shows a schematic view of a breaking pendulum variable-structure model with two modes. One mode which is the normal pendulum and one mode which is a falling mass.

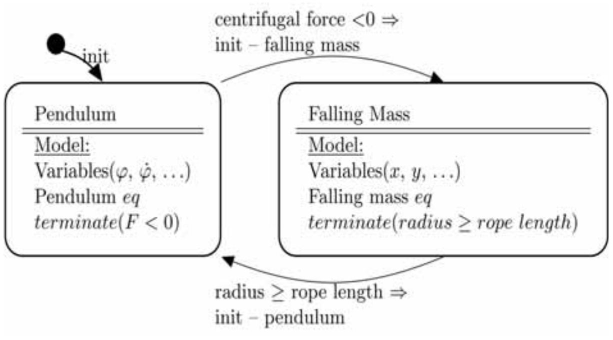

Figure 2: Schematic view of a breaking pendulum variable-behavior model.

To get from one mode to the next a Python script is used, this approach was already presented in [7] where it was tested with different scripting languages. The workflow of the script which handles the simulation of the variable-structure model is shown in Figure 3. In our package the modeled ModelObject is used as input for the variable-structure simulation method.

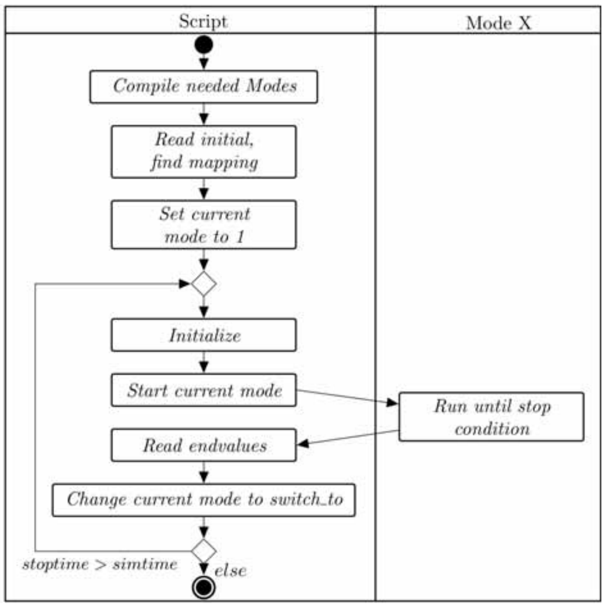

Figure 3: Program flow of the `switch.py` method.

At the beginning of this method the modes are compiled, which results in having an executable called 'dymosim.exe' and an initialization file called 'dsin.txt'. The dymosim.exe can be used to start the simulation of the model. Each created init file is loaded and results in having an initialisation matrix. An identical matrix can be loaded after a simulation of a Dymola model whereas this matrix then holds the end data of the simulation. To make the mode switches faster, the mapping for setting initial values in the init matrix through the end values of the end matrix, is saved in each transition. For now only a one to one mapping is allowed (`oldMode.x = newMode.z` is allowed, `oldMode.z = f(oldMode.x,oldMode.y,...)` is planned) which makes the mapping simple. This of course means that all values necessary to fully initialize the new mode need to be available in the old mode. In case a value is not available the modeler can set values himself in the package. The initialization routine of the specific simulation environment of the mode is then used to initialize the whole model. The modeler is therefore responsible to specify the initialization within the package to get a stable and continuous solution. The method then enters a while-loop which only stops when the user defined stop time is reached. The loop starts with the user defined start mode with given initial values. When the simulation stops because of a specific stop condition the transition to the next mode is known through the ModelObject. The end values of the simulation (dsfinal.txt) are then loaded which gives the end value matrix. The simulation data of the variables to observe are saved in a result matrix. The new mode is then used as current mode and the while-loop is entered again. The mapping which was saved in the transition is then used to set the initial values of the current mode.

If the stop time of the simulation is reached the while-loop is not entered again. After the simulation is done the observed values are saved in a data-file which can later on be used to post-process the simulation data.

1.3 Creating variable-structure models

As an example on how the package can be used, we look again at the pendulum model. First lets consider the Modelica models needed for the variable-structure model. The package approach was chosen because we wanted to be able to use existing models and to reuse the models afterwards again. The package allows us to use our old models on its own because they are valid models, but each model needs a stop condition to be used as a mode in a variable-structure model so the model needs to be altered. Modelica with its object-oriented approach Modelica2010 helps us to keep our old models as they are and extend our needed modes from the old models. Figure 4 shows the pendulum model.

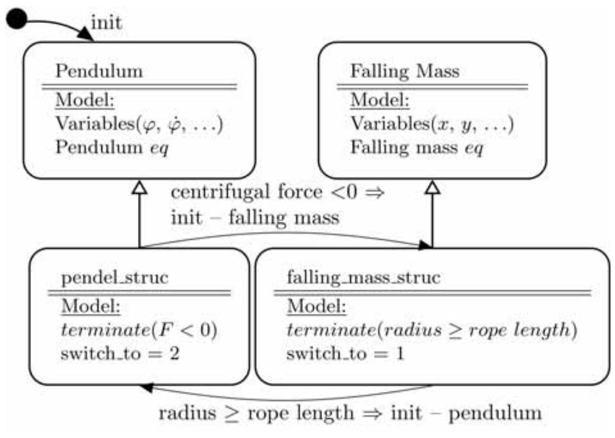

Figure 4: Using inheritance for variable-structure modeling.

The two models 'Pendulum' and 'Falling mass' represent the original models. The other models are extended models of the two and have the stop condition (here called terminate) added and a variable ‘switch_to’ which specifies the mode that needs to be entered next. Here it can be seen, that the original models have not changed and can be used as before only the extended models now represent the modes in the variable-structure model .

After the models for the modes are defined the template provided in the package is used to define the mode switches for the variable-structure model .

```
stop = 10 # stoptime

model = [’Pendelum.mo’] # filename

mode1=’pendel_struc’ #first mode
mode2=’falling_mass_struc’ #second mode
modes=[mode1, mode2] #list of modes
sol=EULER # global solver

# SWITCH MODE 1 - > MODE 2
out1=[’x’,’y’,’der(x)’,’der(y)’]
in2=[’x’,’y’,’vx’,’vy’]
transition.append([1,2,out1,in2])

# SWITCH MODE 2 - > MODE 1
out2=[’x’,’der(phi)’]
in1=[’x’,’dphi’]
transition.append([2,1,out2,in1])

obs([’x’,’y’],[’x’,’y’])

switch(stop,sol,model,modes,transition,obs)
```

For each model the name of the modelfile (or files) and the name of the modes have to be specified. These modes will later be compiled with Dymola. Afterwards the transitions have to be defined. A user defines a transition with the mode numbers of the two modes between which the transition is. Furthermore, the variables to read from the old mode (out1 and out2) and the variables that will be set in the new mode (in2 and in1) have to be specified. The variable ‘observer’ defines which variables should be saved in a data matrix at the end of the simulation. The data of each mode is mapped and saved in one data matrix (for instance: mode1.x and mode2.x will be written in one column of the data matrix). The data matrix is per default saved as MAT-File but the user can specify other output filetypes as well. All the information is then given to the switch method which creates the ModelObject and all the mode objects with their transitions. This template can be used to specify an arbitrary number of modes and switches between these modes.

2 Evaluation

In this section we present different variable-structure models. With these models it is shows how useful variable-structure models are. We then use an easy variable-behavior model to analyze the scalability of the Python package. At the end of this section requirements a model needs to fulfill to be used as a mode in a variable-structure model are discussed.

2.1 Variable-detail models

To show that variable-detail models can save simulation time we look at a diesel combustion engine model, see Figure 5. In this model the environment pressure changes every five seconds and thus the pressure of the manifold changes. When the pressure of the manifold and environment are almost the same the throttle and manifold are not necessary anymore and can be taken out. When the pressure changes again the model has to become more detailed again to simulate the dynamic pressure change in the manifold.

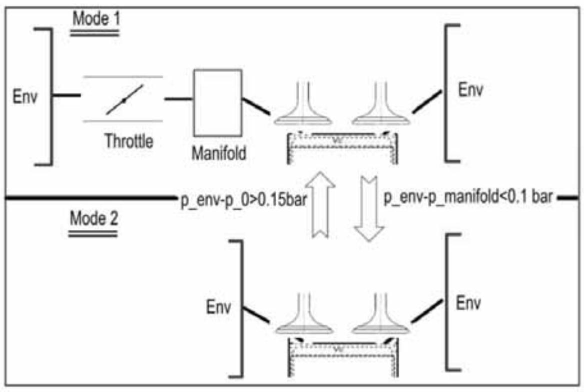

Figure 5: Schematic view of a diesel combustion engine variable-detail model.

The measured simulation times can be seen in table 1. Here the simulation times of the model with only one level of detail and with two levels of detail are presented. The model is always simulated for 20 seconds and has 7 mode switches. The *Compilationtime* is the time needed to compile the Dymola models and the *Residualtime* is the time the script needs for starting the simulation, setting initial values and so on. The variable-detail model takes less time than the one level of detail model even though two compilations are necessary and the residual time is larger. This leads to the conclusion that variable-detail models can make a simulation faster. We also compared the results of the cylinder pressure and temperature and the difference was less than half a percent which shows that we were able to make the simulation faster without significant loss of accuracy.

Stop time 20sec /7 switches	**One detail Dymola**	**Variable detail Dymola/Python**
Simulation time	17	7.3
Compilation time	1*1.2 = 1.2	2*1.2 = 2.4
Residual time	1	2.3
Total time	**19.2**	**12**

Table 1: Simulation time for the diesel combustion engine in seconds.

2.2 Variable-behavior models

As an easy variable-behavior model we present a bouncing ball model. Here the bouncing ball does not just change its velocity when it hits the floor but becomes a spring and damper system which means the ball is elastic and bounces differently depending on the damping constant. Figure 6 shows the bouncing ball results with different damping constants. Interesting is that the ball can never fall below the surface as happens if only a 'when' statement n which the velocity is negated and multiplied by a factor (without extra precautions) is used in Modelica.

Furthermore, the deformation of the ball can now be modeled dependent on the current velocity of the ball, which makes the model more realistic. As a model with more than two transitions we present a breaking pendulum model where the rope can get stuck on a nail. The model is simplified to make it easier to understand:

- The pendulums suspension point is (0,0)
- The nail position is $x \leq 0$ and $y < 0$ ('nailPoint')
- The falling mass model is valid left of the nail

The simplified model is shown in Figure 7 (only a few important variables are shown in this view).
We still have two modes but when the rope passes the angle where the nail is located the rope length changes and therefore the suspension point of the pendulum. We make a mode switch into the same model but change the model parameters. If the centrifugal force goes below

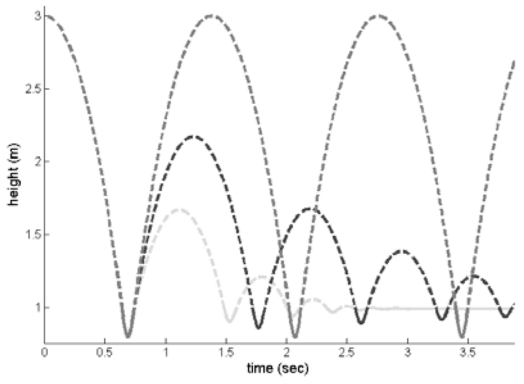

Figure 6: Simulation result of the center point of the bouncing ball variable-structure model .

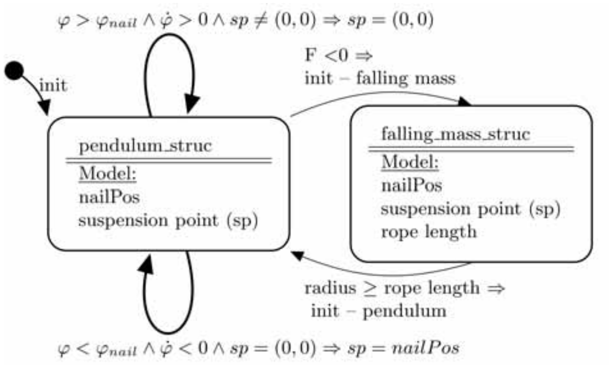
$\varphi > \varphi_{nail} \wedge \dot{\varphi} > 0 \wedge sp \neq (0,0) \Rightarrow sp = (0,0)$
init
F <0 ⇒
init – falling mass
pendulum_struc
Model:
nailPos
suspension point (sp)
falling_mass_struc
Model:
nailPos
suspension point (sp)
rope length
radius ≥ rope length ⇒
init – pendulum
$\varphi < \varphi_{nail} \wedge \dot{\varphi} < 0 \wedge sp = (0,0) \Rightarrow sp = nailPos$

Figure 7: Constrained and breaking pendulum variable-structure model .

zero the pendulum becomes a falling mass otherwise it either turns around the nail or becomes the normal pendulum again. Figure 8 shows different movements of the pendulum for different start values of the angular velocity (dphi (rad/sec)) and the damping constant (D (N sec/m)). The normal suspension point and the nail are markedy as dots.

2.3 Scalability of the simulations

The scalability is always an issue with programs as presented here especially if one goal is to save simulation time. To test if the scripting approach with Python scales with the number of switches and the number of variables for initialization two different tests were made. For both test a bouncing ball model is used which has a transition to itself as soon as the ball touches the ground. The velocity is then negated and we have a ball that never stops bouncing.

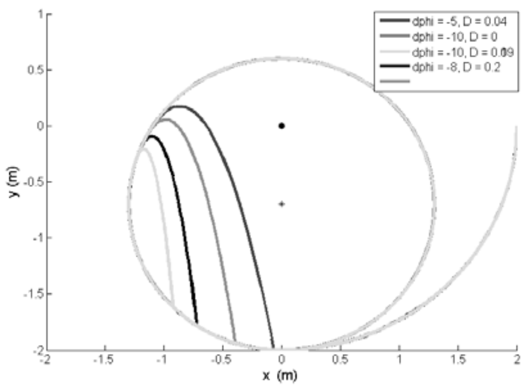

Figure 8: Simulation results of the breaking pendulum model with nail.

This model has only 2 statevariables (height,velocity) and these have to be initialized for each mode switch. As first test this model is simulated with 10, 100, 1000, 10000 mode switches, see table 2. It can be seen that the simulation time scales with the number of switches and it can also be seen where most of the time is lost.

Switch	overall	mapp	while-loop			
	time	time	CPU	dymosim	read	init
10	0,64	0,046	0,02	0,33	0,22	0,02
100	5,21	0,039	0,10	2,90	1,98	0,17
1000	55,34	0,049	0,68	30,38	22,35	1,79
10000	544	0,039	6,20	298,00	220,22	18,33

Table 2: Simulation time for many switches in seconds.

The start of the dymosim.exe takes up a long time, but we do not have any means to change anything on the dymosim.exe routine (each time the executable is started the license is checked, which also takes up time). The other part that takes a long time is the reading of the end values of the old mode. In the implementation the dsinfinal.txt (which holds the end values of the simuation) is changed into a Matlab file, because it is easier to load. This process does take up a long time and we are currently trying to find a better solution. The other measured times are the CPU time which is the time the simulation runs, the init time is the time it takes to set the initial values in the initial file. This of course is a rather drastic example because the idea of variable-structure models is to use larger models with a long CPU time and a few switches and not a model with almost no CPU time and many switches.

As second example we use the same bouncing ball but this time we create an array with many of these balls. We always simulate for 10 switches but with 10, 100, 1000, and 10000 bouncing balls. This leads to many statevariables which have to be initialized. Now we see in table 3 that the CPU time takes up most of the time, which was to be expected from larger models. We see that finding the mapping at the beginning of the simulation takes up a long time. There it can be seen that it is feasible to search the mapping once at the beginning and not for each switch because the needed time would be even greater. All other measured times are rather insignificant compared to the large CPU time.
We see here that there is still some improvement necessary for the index search but otherwise the approach seems good for large models.

Balls	overall	mapp	while- loop			
	time	time	CPU	dymosim	read	init
10	5,00	0,06	0,4	4,3	0,17	0,02
100	6,75	0,07	0,23	5,5	0,24	0,45
1000	14,93	2,4	7,4	4,03	0,21	0,61
10000	322,27	62,15	252,2	5,47	0,77	0,85

Table 3: Simulation time for large models in seconds.

2.4 Model requirements

After introducing the Python package, its design, and presenting examples of variable-structure models we are now discussing the most important requirements for variable-structure models with our Python package.

Looking at the scalability test it is clear that it is not feasible to create variable-structure models with lots of mode switches especially if the models them self are really small. Many switches lead to an overhead in simulation time through the scripting. For variable-behavior models this might still be reasonable because one might otherwise not be able to simulate the system at all.

Another problem with many mode switches is the initialization of the new mode. Each time a switch occurs an initialization problem has to be solved. If the values are chosen incorrectly or cannot be calculated from the old mode the numerical solution might become wrong or even instable. The initialization is therefore a great issue for variable-structure models and it is only possible to switch from one mode to the next if the new modes statevariables can be calculated through the variables of the old mode. This means not all models are fit to be used as modes in variable-structure models.

For variable-detail models it is important that the models have a CPU time which is greater than the time the script consumes and also that the less detailed model at least compensates the scripting time otherwise no simulation time can be saved.

3 Conclusion and Future Work

Our approach is not able to re-causalize only the needed equations or to have a just-in-time compiler as some other language and tools for variable-structure models have but we are able to use a common simulation environment for our simulation and therefore use the strength of this tool.

We can use a tool like Dymola and give modelers the opportunity to test if variable-structure models are feasible for them.

Our approach helps to easily create variable-structure models from existing models and use easy means to describe the models. We therefore hope that with our Python package knowledge of variable-structure models can be gained.

In the future the package will be enhanced to a framework which will support different simulation tools and a graphical user interface. The framework should enable the user to use models from different tools as modes.

With the planned framework researches are planned on what a tool needs to be usable for variable-structure modeling and when variable-structure models should be used to be feasible.

References

[1] MATLAB/Simulink Release 2010b, The MathWorks, Inc., Natick, Massachusetts, United States.

[2] dynasim: Dymola [Internet]. Dassault Systèmes c2002-2014 [cited 2014 Dec]. Available from: `www.dynasim.se`

[3] Nytsch-Geusen C., Ernst T., Nordwig A. et al. Mosilab: Development of a modelica based generic simulation tool supporting model structural dynamics. In G. Schmitz, editor. *Proceedings of the 4th International Modelica Conference*; 2005 Mar, TU Hamburg, 527–535.

[4] Zimmer, D. *Equation-Based Modeling of Variable-Structure Systems* [dissertation]. [Swiss Federal Institute of Technology (CH)]. ETH Zürich; 2010.

[5] Nilsson H, Giorgidze G. Exploiting structural dynamism in Functional Hybrid Modelling for simulation of ideal diodes. *Proceedings of the 7th EUROSIM Congress on Modelling and Simulation*; 2010; Prague: Czech Technical University Publishing House.

[6] Urquia A., Dormido, S. Object-oriented description of hybrid dynamic systems of variable structure. *Simulation*. 2003; 79(9): 485–493.

[7] Mehlhase, A. Varying the level of detail during simulation. *ASIM 2011, Symposium Simulationstechnik*; 2011.

[8] Elmqvist H, Cellier F.E., Otter, M. Object-oriented modeling of hybrid systems. *Proc. 1993 European Simulation Symposium*; 1993; Delft.

[9] MOSILAB [Internet]. 2011 [cited 2014 Dec]. Available from: `http://mosim.swt.tu-berlin.de/wiki/doku.php?id=projects:mosilab:home`

[10] Nordwig, A. *Integration von Sichten für die objektorientierte Modellierung hybrider Systeme* [dissertation]. [Institut für Softwaretechnik und Theoretische Informatik (DE)]. Technische Universität Berlin; 2003.