Simulating a Variable-structure Model of an Electric Vehicle for Battery Life Estimation Using Modelica/Dymola and Python

Moritz Stüber[1]

[1]University of Applied Sciences Vorarlberg, Austria

Abstract

A variable-structure model (VSM) of a battery electric vehicle used for simulating the ageing of the battery pack is presented. The operating principle of the software used to simulate the models is described and a brief summary of the state of science and technology regarding the simulation of VSMs is given. By comparing the performance of the VSM to a conventional model, it is found that the simulation time does not necessarily decrease when replacing a model with a variable-structure version. However, the VSM has advantages regarding the handling of the result files and the possibility to analyse the results.

Keywords: Variable-structure Model, VSM, Modelica, Dymola, Simulation

1 Introduction

In recent years, the use of electrified or electric power trains in passenger cars has gained renewed research interest. However, most currently available battery electric vehicles (BEVs) have a rather low driving range caused by the low energy density of the battery pack in comparison to conventional fuel. The battery pack is not only the single most expensive part of the BEV, but also subject to significant degradation. Consequently, car manufacturers have to simulate the state of health (SOH) of the battery for two main reasons: on the one hand, it has to be made sure that a vehicle still meets the requirements when the battery has aged. On the other hand, it is necessary to estimate possible warranty costs caused by battery packs that reach the end of their life ahead of time.

With age, the capacity of the battery decreases and the impedance increases. Since this affects the electrical quantities within the vehicle, there should be no separation between the model used for simulating the driving behaviour and the ageing model. Instead, the model should combine electric, mechanical and thermal models and thus be capable of calculating feedback effects. Due to the fact that the battery life has to be simulated for 8 to 15 years, simulating this complex model takes a substantial amount of time. Therefore, a speed-up of the simulation is desirable.

Vehicles are idle for the majority of their life. During this time, the complexity of the model used for simulation can be much lower than during driving. Implementing this change in the level of detail of the model leads to a so-called variable-structure model (VSM). Simulating a VSM is potentially faster and/or more accurate than a conventional simulation, but VSMs are not yet supported by commonly used modeling languages and simulation environments.

In this paper, the results of investigating the question "Does the simulation time of the ageing process of a battery used in electric vehicles decrease if the system is systematically modeled as a variable-structure model?" (Stüber 2016) are presented.

In order to answer this question, four steps were taken. First, a conventional model capable of estimating the battery life of a BEV was assembled using Modelica/Dymola. Then, a variable-structure version of this model was implemented and a software capable of simulating the VSM using Dymola's Python interface was written. Last, a series of simulations was performed in order to investigate the influence of the model and solver settings on the time needed to execute the simulation.

In the next section, a brief introduction to the modeling and simulation of variable-structure models is given.

2 Variable-structure Models

In the context of modeling and simulation, variable-structure models are models that consist of *several* sets of equations describing the same physical system. Each set of equations is called a "mode" of the model; exactly one mode is active at all times during the simulation. The changes between the modes are denoted "transitions". For each mode, the transitions define which mode becomes active next, the condition c_i for changing and information i_j on how to initialize the next mode (Mehlhase 2015, chapter 3.2)—compare Figure 1.

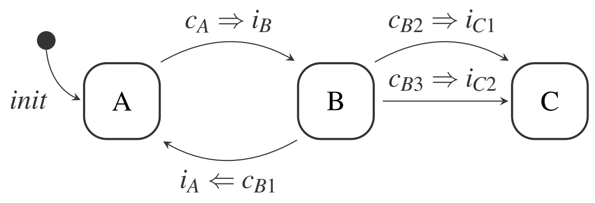

Figure 1. Graphical representation of a VSM with three modes A, B and C

As the name suggests, variable-structure models exhibit varying *structural* properties, which can either refer to the properties of a real system or to the properties of the system of equations used to mathematically describe it.

In *variable-structure systems (VSSs)*, the change in structure is a property of the physical system. Failure situations like breaking mechanical or electrical connections or so-called agent-based systems represent examples of VSSs.

On the other hand, in *variable-structure models (VSMs)*, the change in structure is a result of abstraction; in other words the result of *creating a model* of a system. For example, if detailed information about the switching process is not relevant for an experiment, ideal switches are used. Since ideal switches can attach or detach whole parts of a model, a change in the structure of the underlying set of equations occurs: variables and relations can change and the system of equations can grow or shrink in size.

Variable-structure models can be used to implement changes in the behaviour or the required level of detail of the system under investigation. Components can be added and removed during the simulation. This is necessary for simulating agent-based systems or changing the discretization of a model by changing the number of identical components; as well as for implementing ideal switches, breaking connections or limiters, leading to the dynamic addition or removal of parts of the model (Mehlhase 2015, chapter 4; Zimmer 2010, chapter 1.2). Furthermore, changing the solver and the solver settings during a simulation is possible when simulating VSMs.

Despite the multitude of use cases for VSMs, only very few simulation environments support their definition and execution. According to Zimmer (2010, chapter 1.3), there are two main reasons for this: first, current modeling languages lack the expressiveness required to accurately define the structural variability. Second, it is technically very challenging to simulate the resulting models.

Three different concepts have been developed for implementing a simulation engine that can handle variable-structure models: maximal state-space, hybrid decomposition and dynamic causalization.

- In a *maximal state-space*, “state events switch on and off algebraic conditions, which freeze certain states for certain periods” (Breitenecker 2008, page 9). The maximal state-space-model is static and can therefore be simulated using conventional tools.

- In contrast, when using the *hybrid decomposition*-approach, the VSM is split into its modes, which have a static structure and can be executed sequentially. The order of execution is controlled at a metalevel, either within the simulation environment or externally.

- The most flexible, but also the most challenging approach from a technical point of view, is called *dynamic causalization*. Here, the model is re-causalized if necessary, which is impossible when using the usual translation–compilation–execution sequence.

Because structural changes *always* cause events and VSMs thus represent a generalization of hybrid models, modeling languages that support them need to provide a generalized way to *define* events in order to achieve the required expressiveness.

The most recent, Modelica-based[1] attempts to implement a modeling language and a corresponding simulation environment that support the simulation of VSMs are Sol, DySMo and MoVasE as well as an unreleased prototype of Dymola.

Sol Sol (Zimmer 2010) is an experimental language which is intended to serve as a proof of concept for simulating variable-structure models using dynamic causalization. Conditional index changes as well as the local definition of modes within components are supported. Sol therefore allows the modeling and simulation of “almost arbitrary structural changes” in a truly object-oriented manner (Zimmer 2013), but the proposed language constructs and simulation techniques have not been integrated into commonly used languages and tools yet.

Dymola Elmqvist, Mattsson, and Otter (2014) presented an approach to simulate VSMs in Dymola. It represents an extension to the capabilities of the synchronous state machines defined in Modelica 3.3 and was implemented in a prototype version of Dymola 2015. Instead of defining additional language elements, the semantics of the existing language was extended.

Using this approach, a large, but limited class of VSMs could be simulated. Because it is not necessary to process the model definition using an interpreter, like in Sol, the simulation is significantly faster. However, variable-structure models with varying index could not be simulated. This possibility was added later by extending the Pantelides algorithm (Mattsson, Otter, and Elmqvist 2015), but it has not been added to the official version of Dymola yet.

DySMo (**Dy**namic **S**tructure **Mo**deling) is a Python application that allows the simulation of VSMs (Mehlhase 2015, chapter 7). Each mode is represented by an executable model with static structure that terminates if the condition for a transition is triggered. Upon termination, a variable is set that defines

[1]Tools that support VSMs to a certain degree, but rely on different modeling concepts have been developed outside the context of simulating physical systems, for example Hydra (functional programming), JAMES (systems biology) and ANYLOGIC (large-scale agent-based systems). They are not suited for simulating the BEV model and thus not considered further.

the cause for the transition. DySMo then reads and stores the results, initializes the next mode depending on the cause for the transition and starts the next simulation. All modes and transitions have to be maintained manually.

MoVasE Esperon, Mehlhase, and Karbe (2015) propose a methodology to append structural changes to existing models by externally defining conditional component exchanges. The tool MoVasE (**Mo**delica **Va**riable-**s**tructure **E**ditor) implements the proposed solution. The aim of MoVasE is to provide a platform for facilitating the investigation of VSM-design. In contrast to DySMo, MoVasE does not require the user to create and maintain all modes manually. By defining the structural variability through conditional component exchanges, many modes can be created and maintained. However, the flexibility of this approach is still limited with regard to the dynamic addition and removal of components.

In conclusion, the approaches taken by Sol and Dymola solve the most important technical problems, but they have not been integrated into standard languages and tools yet. Therefore, script-based approaches (DySMo, MoVasE) are necessary for studying the benefits and drawbacks of *using* variable-structure models.

Three levels of complexity can be distinguished when *creating* VSMs: in the simplest case, the modes are defined on the highest level of the model, as shown in Figure 1. Second, individual components of the model can exhibit modes. In the most complex case, the modes are a result of the addition and removal of components, like in agent-based systems.

From the point of view of the simulation engine, a variable-structure model *always* consists of modes defined at the highest level of the model, which is called the *factorized* version of a VSM (Mehlhase 2015, chapter 5.1.2). Depending on the tool used for simulation, it might be necessary to manually create the factorized version of the VSM. Additionally, it has to be made sure that the VSM is *valid*. This includes avoiding chattering or unphysical transitions and unphysical factorized modes. A set of guidelines that is intended to help with the creation of valid VSMs was formulated by Mehlhase, Esperon, and Karbe (2015).

In addition to many small examples that were used to verify a certain language/tool (Zimmer 2010, chapter 11; Mehlhase 2015, chapter 8), several examples of the successful application of variable-structure models for solving real-world problems have been published (Krüger, Mehlhase, and Schmitz 2012; Mehlhase, Esperon, Bergmann, et al. 2014; Möckel, Mehlhase, and Nytsch-Geusen 2015). In these examples, a significant reduction of the overall time needed to simulate the system could be achieved due to a big difference in the complexity of the modes and a low number (≤ 40) of mode switches.

So far, no attempt to use a VSM of a BEV for estimating its battery life has been published. In the next section, the models used by the author are described, followed by a description of the observed advantages and disadvantages.

3 Implementation

In order to assess the usefulness of using a VSM of a BEV for battery life estimation, both a conventional and a variable-structure model were assembled and simulated. Implementing the *components* used for assembling the models was *not* part of this work; they are part of the commercial *Electrified Powertrains Library* and the *Battery Library* developed by Dassault Systèmes[2].

The conventional model consists of nine main components: the driving cycle, the driver model, a control unit, the models of the energy supply, the electric power train, the auxiliary loads, the chassis and the environment, as well as the charger model. In contrast, the VSM has two global modes that reflect the "operating modes" of the vehicle (Figure 2). The model that represents the "driving" mode does not contain the charger and its control logic, while the model that represents the "idle" mode *only* contains the battery model, the charger model and the environment model.

The inputs of both models are the desired speed of the vehicle, the time frames the charger is plugged in, and the ambient temperature over time. The VSM additionally has a schedule for switching between modes based on the driving behaviour. All inputs are loaded from externally stored files which are generated using a script. The script allows the convenient definition of usage scenarios and ensures that the resulting profiles are consistent.

The system of differential algebraic equations (DAEs) of mode "idle" has 915 scalar unknowns and equations and 25 continuous time states. Mode "driving" consists of 3062 scalar unknowns and equations and has 29 continuous time states; the conventional model of the BEV has 3219 scalar unknowns and equations and 38 continuous time states. Since the system of ordinary differential equations (ODEs) that needs to be integrated differs only slightly, it can be expected that the speed-up is small, but noticeable. It would be straightforward to use more complex models of the power train during driving because a template is used that allows the simple exchange of models. This would likely increase the performance gain of the VSM compared to a conventional model, but also increase the overall simulation time in both cases.

The VSM is simulated using a software written in Python that allows the definition and simulation of VSMs with globally defined modes using Modelica and Dymola. The software is comparable to DySMo and described in section 6.

[2] `http://org-www.3ds.com/products-services/catia/products/dymola/industry-solutions/`

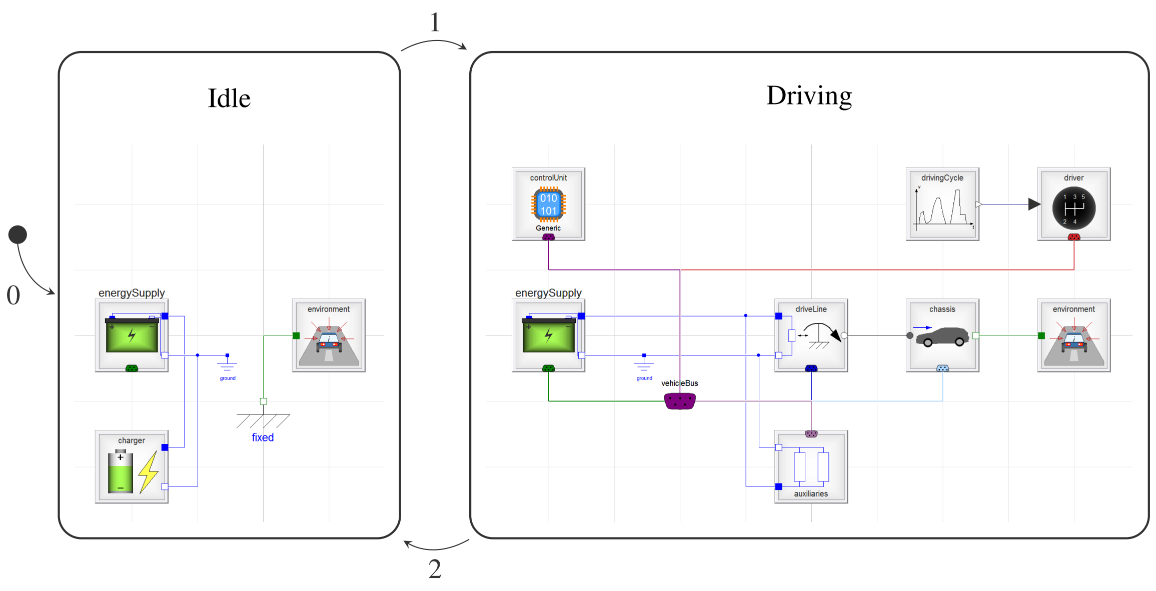

Figure 2. Variable-structure model of the BEV

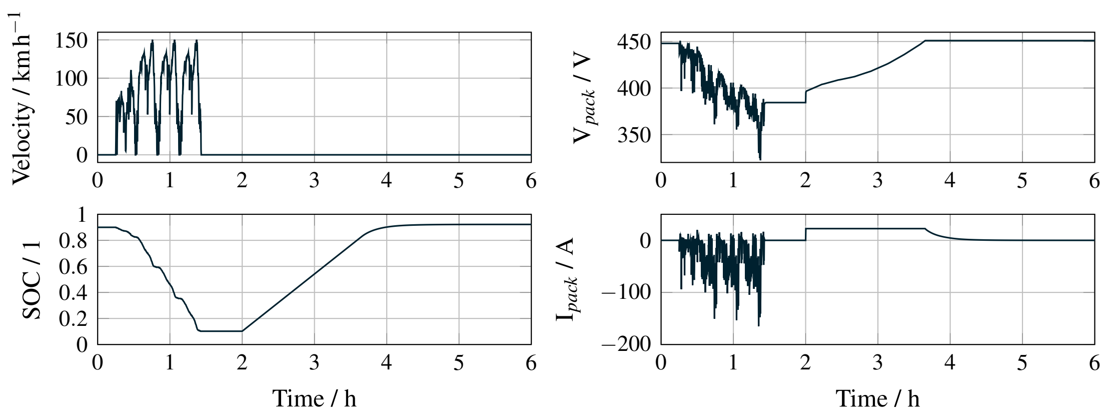

Figure 3. Selected results of the experiment used for model validation

4 Simulation Results

The parameters of the models were chosen with the intention to reflect typical values for BEVs. The models were checked for plausibility by simulating a short driving cycle followed by charging. The driving phase starts after 15 min standstill and takes approximately 70 min. During this time, a distance of 103 km is covered and the battery is discharged from 90 % state of charge (SOC) to 10 % SOC, which corresponds to a total consumed energy of approximately 16.2 kWh. About half an hour after the driving cycle is completed (at $t = 2\,\mathrm{h}$), the charger is plugged in. In Figure 3, selected results of the simulation can be seen. There is no noticeable difference between the results of the conventional simulation and the results of the VSM.

Knowing that the models are properly parameterized, a simulation of the SOH spanning several years could be performed. Two usage profiles were created, which are shown in Figure 4 and Figure 5. Both span a week and are used repeatedly if longer scenarios are needed. In the first scenario, the battery is always charged fully, whereas in the second profile, the SOC only varies from approximately 60 % to 40 %.

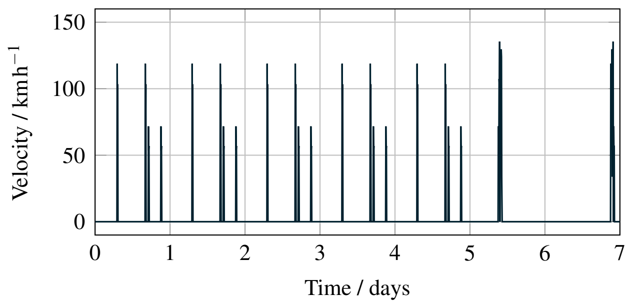

Figure 4. Demanding usage scenario: desired velocity. When using this driving cycle, 658 km are driven per week; the yearly mileage amounts to 34 238 km.

The higher usage and the storage at higher SOC should result in faster ageing of the battery in the first scenario.

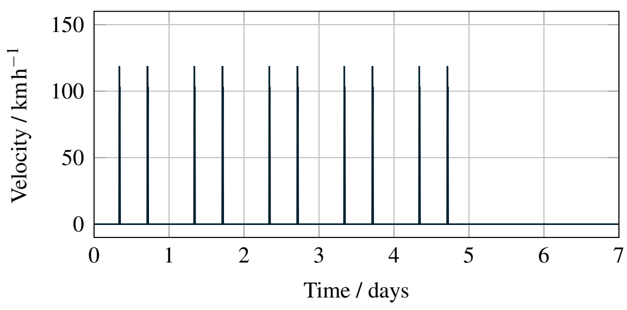

Figure 5. Relaxed usage scenario: desired velocity. When using this driving cycle, 289 km are driven per week; the yearly mileage amounts to 15 040 km.

The simulation results are shown in Figure 6.

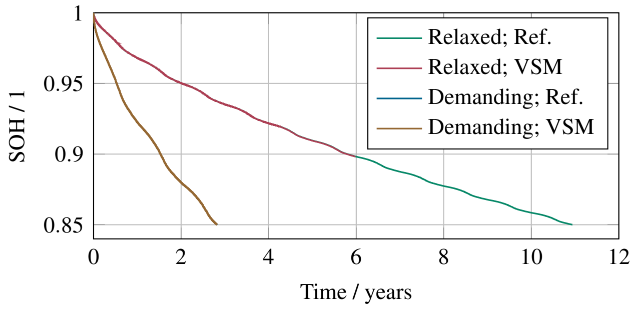

Figure 6. State of health of the battery

For the assessment of the battery's ageing behaviour, the models were simulated until the SOH falls below 0.85. This value was chosen because a compromise between the time needed for simulation and the length of the calculated trajectory had to be found and a ΔSOH of 0.15 is regarded meaningful. Moreover, in this case, more data does not mean more information because all input and model parameters are estimated values anyway.

There is no significant difference between the result of the conventional model and the VSM: the relative error is in the range of ±0.02 %.

5 Advantages and Disadvantages of the Variable-structure Model

In Figure 7 and Figure 8, the elapsed CPU time during a 2-week simulation of the BEV model using the "relaxed" driving cycle (Figure 5) is shown. The CPU time comprises the time needed for initializing the system(s) of equations and the time needed for integration. The same solver settings are used for both the conventional and the variable-structure model.

In both figures, the difference between driving and standing can be seen clearly due to the step wise increase of the elapsed time resembling a staircase. This is the result of using a solver with variable step size: during driving, only a small step size can be used due to the dynamic changes of the state variables. In contrast, the step size can be greatly increased when the vehicle is idle.

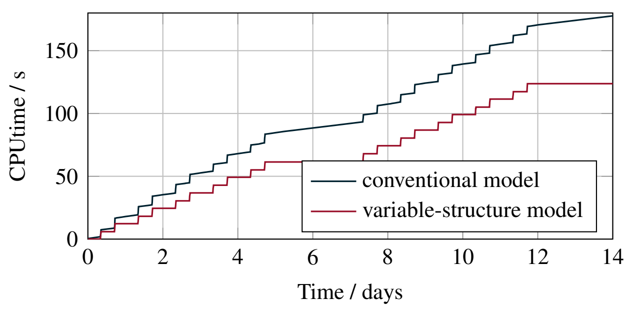

Figure 7. CPU time; dense output enabled

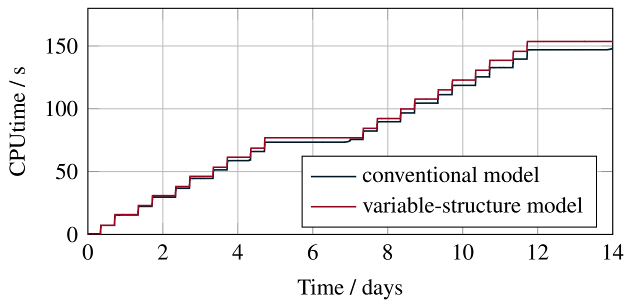

Figure 8. CPU time; dense output disabled

There is a significant difference in the elapsed time depending on whether dense output is enabled or not. If dense output is enabled, the conventional model takes longer to calculate, depending on the size of the output interval. The effect is especially noticeable during the idle phases because during this phase, the *necessary* step size calculated by the step size control algorithm is usually *bigger* than the desired output interval.

However, when dense output is disabled, the conventional simulation becomes *faster* than the variable-structure model. A reason for this is that each system of equations (mode) needs to be *initialized*.

In Figure 9, a more detailed account of how much time is spent on which part of the simulation is given. On the right hand side, two pie charts visualize the data listed in the table on the left. The area of the pie charts corresponds to the total duration of the simulation, whereas the slices denote the time spent on the initialization of the systems of equations, the integration itself and the post-processing of the data. Additionally, time is needed for writing the result files and "consumed" by the operating system for other processes. There is a striking difference between the conventional and the variable-structure model: in the former, the integration takes 99.9 % of the time, but only 52.2 % of the time needed for simulating the latter is actually spent on the integration. Therefore, the VSM takes longer to simulate, even though the integration finishes 0.5 h earlier. A large, *unnecessary* part of the overhead is caused by Dymola's Python interface: since the `simulateExtendedModel()`-command only

Task	*Reference*	*VSM*
Initial Compilation	4.5 s	7.4 s
Initialization	1.3 s	1.0 h
Integration	7.7 h	7.2 h
Recompilation	—	5.6 h
Reading .mat-files	≈ 0.0 s	19.3 s
Post-processing	≈ 0.0 s	1.2 h
Total Duration	7.7 h	15.1 h

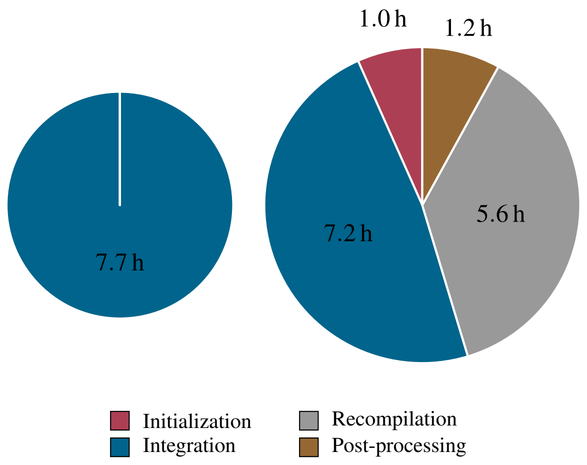

Figure 9. Comparison reference model–VSM: time measurements for a simulation of the demanding scenario for 3 years. In the variable-structure model, 5304 mode switches were performed.

supports passing real numbers for initialization, modifiers have to be used for passing the necessary vectors and attributes. This causes a recompilation of the model at each mode switch. By finding a workaround for this, the overall simulation time of the VSM could be reduced drastically. A further reduction could be achieved by improving the implementation of post-processing.

When working with VSMs, besides the restriction to use the same *model* for all phases, also the restrictions to use the same *settings* and *result files* for all phases no longer exist. Therefore, the question "Does it *make sense* to use a variable-structure model of a BEV for estimating its battery life?" may still be answered with "yes", even if the time needed to calculate the output trajectory does *not* decrease very much.

One problem that arises when estimating the battery life using a full vehicle model is the size of the result file, which depends on the settings for dense output and the length of the simulation. In order to limit the amount of data stored in the result file, the output interval needs to be set to a constant, high value (for example 24 h when simulating 15 years). Additionally, it is necessary to select the set of variables that has to be stored in advance, for example by using Dymola's `__Dymola_selections`-annotation. This means that all detailed information calculated during the course of the simulation is irretrievably lost and not available for analysis. When simulating a VSM, this problem is much less likely to occur as every mode has its own result file and the individual simulation times are much shorter[3]. Moreover, it is possible to store the results in a high resolution when the vehicle is driving and in a low resolution otherwise. Therefore, it becomes possible to perform a detailed analysis of the driving behaviour at the end of the battery's life.

[3]Strictly speaking, the memory limitations could also be avoided by implementing the possibility to split up result files when performing a conventional simulation in Dymola.

6 PyVSM

In this section, the software used for implementing and simulating the VSM of the BEV is described. It is called PyVSM and supports the simulation of factorized variable-structure models using Dymola's Python interface. PyVSM is intellectual property of Dassault Systèmes Deutschland GmbH.

The basic idea of PyVSM is to use Dymola for simulating the modes and Python for switching between them. Therefore, it is required that each mode of the *factorized* VSM is a complete Modelica model that can be simulated using Dymola. Modes, transitions and solver settings of the VSM are defined using JSON-files. When a condition of a transition becomes true, the simulation of the currently active mode terminates and the initialization of the next mode is initiated by PyVSM based on the results of the previous mode.

In Figure 10, the processing steps taken to simulate a VSM in PyVSM are shown in more detail. First, the JSON-file used for the definition of the VSM is read. Then, for each mode, a directory used during simulation is created and the Dymola-interface is instantiated with the working directory set to the previously created folder. The actual simulation of the VSM starts by executing the initial transition in order to set the initial values for the simulation. The active mode is set according to the definition of the transition and the simulation is started. Upon termination, the relevant results are loaded to PyVSM, including the variable `transitionID`. Its value corresponds to the numerical identifier of the transition that has to be executed next. If and only if it is 0, the end of the simulation is reached. After this, the individual results of each mode are concatenated and post-processed. This includes the calculation of characteristic values such as the time needed for initialization or the step size and the resampling of the data to the specified interval length. Last, the post-processed data is saved and plots are generated if desired.

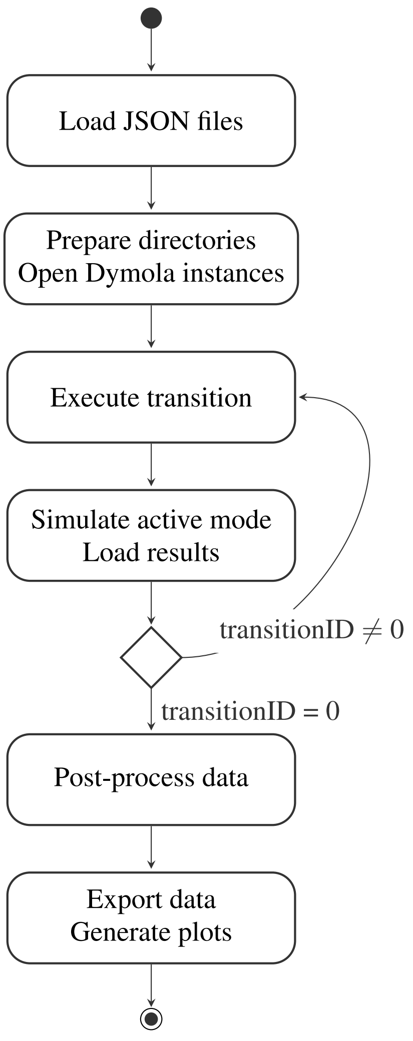

Figure 10. UML-activity diagram of PyVSM

PyVSM is implemented in Python using object-oriented programming techniques. It is capable of simulating factorized variable-structure models using Modelica/Dymola, but, being a prototype, misses advanced features such as automatic validity checks of the input files or a graphical user interface. Additionally, externally controlling the simulation via Dymola's Python interface generates an overhead. Nonetheless, PyVSM provides modelers with the possibility to simulate models that would be difficult or even impossible to implement in a conventional simulation environment in a straightforward manner.

7 Conclusion

A variable-structure model (VSM) of a BEV was implemented with the aim of making the simulation of the battery ageing faster by switching between a complex model used when the vehicle is driving and a simpler model used when the vehicle is idle. Since VSMs are not yet supported by Modelica/Dymola, a software had to be written that provides means to define the structural variability and performs the mode switches.

When simulating the variable-structure BEV model, it is found that while the CPU time needed for integration decreases, thus matching the expectations, the overall simulation time *increases* due to overhead generated by switching between models. This is caused by the properties of the model on the one hand (low difference in the complexity of the modes, many (> 5300) mode switches) and the implementation of the software used for simulation on the other hand. The overhead comprises the excess time needed for compiling, initializing the systems of equations, reading the result files and post-processing them as well as the unnecessary recompilation of the modes at all transitions. Nonetheless, the VSM allows a more detailed analysis of the simulation result due to memory limitations occurring when using a conventional model.

Acknowledgements

This work was supervised by Markus Andres, who contributed by discussing results and giving helpful advice on plans for further work. Marco Keßler and Markus Andres proofread the manuscript. Further help regarding the used battery models and the experiment set-up was provided by Lukas Rohr. The project was carried out during a paid internship at Dassault Systèmes Deutschland GmbH.

References

Breitenecker, Felix (2008). "Development of Simulation Software – from Simple ODE Modelling to Structural Dynamic Systems". In: *Proceedings of the 22nd European Conference on Modelling and Simulation (ECMS 2008)*. DOI: 10.7148/2008-0005-0022.

Elmqvist, Hilding, Sven Erik Mattsson, and Martin Otter (2014). "Modelica extensions for Multi-Mode DAE systems". In: *Proceedings of the 10th International Modelica Conference*. DOI: 10.3384/ecp14096183.

Esperon, Daniel Gomez, Alexandra Mehlhase, and Thomas Karbe (2015). "Appending Variable-structure to Modelica Models (WIP)". In: *Proceedings of the Conference on Summer Computer Simulation*. SummerSim '15. Chicago, Illinois: Society for Computer Simulation International.

Krüger, Imke, Alexandra Mehlhase, and Gerhard Schmitz (2012). "Variable Structure Modeling for Vehicle Refrigeration Applications". In: *Proceedings of the 9th International Modelica Conference*. DOI: 10.3384/ecp12076927.

Mattsson, Sven Erik, Martin Otter, and Hilding Elmqvist (2015). "Multi-Mode DAE Systems with Varying Index". In: *Proceedings of the 11th International Modelica Conference*. DOI: 10.3384/ecp1511889.

Mehlhase, Alexandra (2015). "Konzepte für die Modellierung und Simulation strukturvariabler Modelle". PhD thesis. Technische Universität Berlin, Fakultät IV – Elektrotechnik und Informatik. DOI: 10.14279/depositonce-4514.

Mehlhase, Alexandra, Daniel Gomez Esperon, Julien Bergmann, et al. (2014). “An example of beneficial use of variable-structure modeling to enhance an existing rocket model”. In: *Proceedings of the 10th International Modelica Conference*. DOI: 10.3384/ECP14096707.

Mehlhase, Alexandra, Daniel Gomez Esperon, and Thomas Karbe (2015). “Challenges when Creating Variable-structure Models”. In: *Proceedings of the 5th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pp. 101–110. DOI: 10.5220/0005521601010110.

Möckel, Jens, Alexandra Mehlhase, and Christoph Nytsch-Geusen (2015). “Exploiting Variable-structure Models in the Context of Building Simulations within Modelica”. In: *Proceedings of BS2015*. International Building Performance Simulation Association. URL: https://www.researchgate.net/publication/301229350.

Stüber, Moritz (2016). “Simulating a Variable-structure Model of an Electric Vehicle for Battery Life Estimation Using Modelica/Dymola and Python”. Master’s Thesis. University of Applied Sciences Vorarlberg.

Zimmer, Dirk (2010). “Equation-Based Modeling of Variable-Structure Systems”. PhD thesis. Swiss Federal Institute of Technology, Zürich. DOI: 10.3929/ethz-a-006053740.

– (2013). “A new framework for the simulation of equation-based models with variable structure”. In: *SIMULATION* 89.8, pp. 935–963. DOI: 10.1177/0037549713484077.