A Toolbox for Efficient Parameter and Structure Variation of Time Domain Power System Simulation Models in Simulink

Tilman Wippenbeck, Graduate Student Member IEEE, Reinhold Bertram, Armin Schnettler, Member IEEE
Institute for High Voltage Technology, RWTH Aachen University, Aachen, Germany
stoical@ifht.rwth-aachen.de

Abstract — Variation studies with Matlab/Simulink models are often limited to a handful of variation subjects due to the efforts involved. A toolbox for the efficient automation of parameter and structure variation of complex power system time domain simulation models in Matlab/Simulink has been designed and implemented. A requirement engineering process has been used to define necessary functionality. Several research use cases related to power systems have ensured the usage of best practices in the implementation. The toolbox proves to be beneficial for increasing efficiency in automated parameter and model structure variations and analyses of the results. The authors intend to make the source code and the documentation of the toolbox available for download under an open source license.

Index Terms — MATLAB, Open source software, Power system simulation, Time-domain analysis

I. INTRODUCTION

In time domain computer simulations of electrical power systems model variations are often limited to a few parameters. Those limitations are either due to the effort of performing such or due to restricted computational resources. Increasing availability of the latter enables extensive variations in power system time domain simulation studies.

The need for extensive variation studies increases due to the ongoing penetration of active and controlled components in electrical power systems. Especially the behaviour of generation units with power electronic coupling is dominantly governed by their control rather than physics. Details of the control design (parameters and structure) may result in significantly different reactions to stimuli. Parameter and structure variations need to be performed in order to draw robust conclusions or to design requirements.

Often, control structures and parameters are chosen equally for all similar units. In real world power systems with products of many manufacturers, product families, ages, sizes etc. the resulting diversity should be taken into account, thus making parameter and structure variations necessary.

On the one hand methods for planning such variation studies like design of experiments or sensitivity analysis are well known. On the other hand means for efficiently applying those methods to practical simulation models are lacking. A researcher should be able to focus on his intentions when performing a complex variation study by the help of appropriate tools. A review on variability modelling in industrial environments [1] suggests that some commercial products and a lot of custom made solutions are used, the latter not being available to the public. Within this paper, the authors will focus on solutions for Simulink.

An efficient modelling of structural variants by delta modelling is proposed in [2]. The variation itself needs to be performed by an exogenous tool. A commercial tool [3] focuses on implementing variability management for production oriented use. It is not primarily designed for automation of model structure variations in research tasks. An extension to the tool is proposed in [4] that focuses on aspects of production oriented code generation. Direct scripting of the mechanisms provided by Matlab/Simulink offers a maximum of control. Pragmatically implemented solutions may lack flexibility. Several GUIs provided by Matlab/Simulink are not designed for extensive automation.

Due to the discussed lack, a toolbox for efficient automation of large scale parameter and structure variation of complex power system time domain simulation models has been designed and implemented.

The rest of the paper is outlined as follows. Section II describes the practical challenges that need to be considered in time domain variation studies. Section III presents a solution approach for the Matlab/Simulink simulation environment, which is applied in two examples shown in section IV. Section V discusses benefits and drawbacks. A conclusion is given in section VI.

II. PROBLEM STATEMENT & REQUIREMENTS

When performing variation studies on complex simulation models several practical challenges arise.

A. Input data definition and setting

Numerous design choices need to be made on mechanisms implementing the intended variations. Users may experience a
substantial overhead for implementation, if wrong design choices are discovered late.

A variation study will typically be automated by scripts. Control variables in the model and in the scripts represent active parameter values and model structures. Choosing those control variable names is quickly performed for small models. For complex models with deep model hierarchies with many units and many model parameters or structures that shall be varied, the effort for designing and applying names grows substantially. Names may need to reflect the model hierarchy. Multiple occurrences of variables of similar meaning may need to be distinguished by proper names. Any variable name alterations need to be correctly propagated to scripts and models, usually by hand.

The challenges involved in a faultless usage will often be related to the readability of the scripts. Control variables may not have a readable meaning. Therefore, setting the intended control variables to the correct values to achieve the desired model variation may be error-prone. Faults are often time consuming to track.

Models tend to change in the iterative modelling and research process. Extensive manual changes will need to be undertaken in the model and automation scripts. Hard to track errors may be introduced e.g. by copy paste operations, especially for deep model hierarchies. Model and automation script reuse are generally affected negatively due the effort of manually maintaining integrity.

In conclusion, a solution should:

- choose and implement appropriate mechanisms
- avoid direct user contact with control variables
- offer and ensure clarity of commands for variations
  - make subjects addressable by a meaningful syntax
  - support identical names for identical meanings
- maintain integrity on model changes and evolution
  - allow for copy-paste model extensions
  - support script reuse for existing model parts

B. Output data definition, storage and access

Model output data logging involves design choices on the mechanisms to use. Large scale variations may be infeasible when using false approaches, because storage and access performance become critical for big data volumes.

When accessing results, hardcoded names burden the task of transforming the user’s intention to a correctly functioning script. Especially the comparison of results across structural variants or multiple units may lead to cumbersome scripting.

The association of all relevant input data to the achieved results needs to be maintained. This eases analyses and ensures transparency. Relevant inputs for example include model parameter settings, active model structure variants, and simulation environment and solver configuration.

In conclusion, a solution should:

- choose and implement appropriate mechanisms for signal logging and storage
- avoid direct user contact with internal names
- offer and ensure clarity of access commands
  - make signals addressable by a meaningful syntax
  - let the syntax support identical names for signals of identical meaning
- maintain integrity on model input data and results

III. Solution Approach

A toolbox named “STOICAL” has been designed and implemented for the Matlab/Simulink simulation environment to solve the challenges discussed above. In the following, the toolbox concept, the implementation and the workflow are outlined, whilst section IV gives practical examples of how to use STOICAL.

A. Concept

The functionality desired is decoupled from the enabling mechanisms of the simulation environment by introducing

- a MODEL HIERARCHY,
- ELEMENTS that represent inputs and outputs,
- a SYNTAX for addressing elements in the model hierarchy in a meaningful but unique way and
- ROUTINES acting as an interface and making use of best practice mechanisms for achieving the desired functionality in the majority of use cases.

The model hierarchy represents the model as a directed graph (tree) whose nodes are model units and whose edges represent the relation “contains”. An overlay hierarchy may be defined by the user in order to focus on meaningful units only.

The toolbox defines and uses four types of elements that may be located anywhere in the model:

- STRUCTURES contain a set of antivalent model structure variants.
- PARAMETERS are model input parameters to be set that do not affect the model structure.
- SIGNALS are quantities to be measured and logged.
- CONFIGURATIONS determine solver or toolbox options not represented in block form

B. Implementation

1) Representation of model hierarchy and elements

The STOICAL model hierarchy and elements are represented by using default Simulink means given in Tab. I.

<table>
<thead>
<tr>
<th>STOICAL Entity</th>
<th>Used Simulink technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Hierarchy Units</td>
<td>Subsystems</td>
</tr>
<tr>
<td>Structures</td>
<td>Variant subsystems</td>
</tr>
<tr>
<td>Parameters</td>
<td>Gain blocks, constant blocks, or masked subsystems</td>
</tr>
<tr>
<td>Signals</td>
<td>Signals (defined on lines)</td>
</tr>
<tr>
<td>Configurations</td>
<td>discussed in documentation</td>
</tr>
</tbody>
</table>
2) Declaration of elements

The user declares elements by entering labels into the names of existing blocks or lines of the Simulink model (Tab. I). Labels are strings arbitrarily definable by the user parenthesised by a user definable symbol (e.g. ‘#UNIT#’).

Hierarchical unit, structure, or signal labels are followed by an identifying name (e.g. ‘#LABEL#name’). A list of one or more parameter names can be defined after a parameter label using the defined delimiting symbol.

3) Addressing of elements

Elements are addressed in combination with and in the context of the model hierarchy. Each element is uniquely addressed joining the relevant hierarchy unit names and its own name excluding all labels (e.g. ‘#hierarchy lev1#hierarchy lev2#name’). This is called labelpath in the following. It is thereby possible to use equal names for items of equal meaning. Using regular expressions in the labelpath enables addressing groups of elements efficiently.

Elements map to the next higher hierarchy unit independent of the true model hierarchy. Thereby elements defined deep in the real model hierarchy can be referred to by a short labelpath.

4) Model structure and parameter control

Variants of structures may be activated by routines referring to them using their labelpath and the name of the variant desired. Parameters may be set by routines referring to them using their labelpath and giving the arbitrary value to be set.

5) Log-signal customisation & access

Routines that refer to defined signals by labelpath are provided for

- customising the log status of signals,
- preparing and forwarding the signals after simulation to data storage or direct analysis and
- accessing stored signals during analysis.

Automatic joining of identical signals generated in antivalent variants of structures is supported.

6) Data storage

The toolbox stores all input data, settings etc. influencing the simulation results for look-up during analysis.

Raw result data storage is supported. This allows for continuous redesign and improvement of analysis routines based on gained research knowledge. Usage of a database system has proven to offer acceptable performance for big result data volumes (several hundred GB).

C. Workflow

The described approach leads to the following workflow:

1. model preparation
2. model usage
3. analysis of results

In the model preparation phase all necessary declarations of the hierarchy and elements are performed. STOICAL supports the user in creating valid and consistent declarations. Default parameter values and default variants of structures are defined. After major model changes the preparatory steps need to be redone. In the usage phase, STOICAL needs to parse the model after loading it into memory. The user then sets active parameter values and structure variants, chooses signals to log and other influencing factors like the modelling toolbox or solver configurations by means of routines. After each simulation the user initiates input and result data storage. In the analysis phase the user makes use of interface routines for accessing stored signals and model input data.

Routines and example scripts are provided to support the workflow offering experience based recommendations.

IV. EXEMPLARY APPLICATIONS

A. Illustratory usage example

The electrical distribution system in Fig. 1 shall illustrate the STOICAL toolbox usage and workflow. Several inverter interfaced generators (IIDG) are connected to a feeder consisting of several four-conductor lines. A line fault occurs at the end of the feeder.

![Figure 1. Single-line diagram of example distribution system.](image)

The variations to perform for a fictive research task affect the line models and their parameters, IIDG control structures as well as their reference current amplitude and the fault type. The signals of interest are the voltages and currents injected by the IIDG at their point of common coupling (PCC).

Starting with an existing Simulink model using the SimPowerSystems toolbox (SPS), the workflow from section III.C is used for applying the STOICAL toolbox.

1) Model preparation

A STOICAL model hierarchy is declared by labels (here: #UNIT#) in the names of the top level components. STOICAL structures are introduced for the lines, the fault and the IIDGs (Fig. 2).

![Figure 2. Simulink model representing Fig. 1 with STOICAL hierarchy units and STOICAL structures (overlapping squares symbol).](image)

For example, the structure for the fault is introduced by entering it into a Simulink variant subsystem and creating a variant for each type of fault (one-, two-, or three-phased). Adding a #DEF# label to a variant name indicates the default variant (Fig. 3).
A multi-level hierarchy of STOICAL structures is introduced in case of the IIDG assuming several top level variants implement control structures in different reference frames. In one of the variants, the phase-locked loop (PLL) of the IIDG is introduced as an additional hierarchy unit. Furthermore, a STOICAL structure is introduced whose variants represent two approaches (Fig. 4).

STOICAL parameters are now introduced to the line model variants. The default SPS line model is included in a user defined masked subsystems (UDMS). The STOICAL parameters are declared in the name of the UDMS (Fig. 5). According mask parameters are introduced in the UDMS. Those mask parameter names are entered in the SPS line model. Such embedding needs only to be performed once as the result may simply be copied afterwards or may be placed in a library.

STOICAL signals are introduced to log the IIDG voltages and injected currents by entering a label in the corresponding Simulink line names (Fig. 6).

The model may now be extended by further modelled lines and IIDGs by simple copy-paste operations and connection of the new blocks. As STOICAL handles all names and variables for the mechanisms of Tab. I internally, duplicate name conflicts of conventional Simulink Signal names, ToWorkspace or ToFile blocks are successfully avoided.

Scripts for visualisation of the declarations and consistency checks are delivered with the toolbox. The declarations are finally registered to and processed by STOICAL with the help of several provided scripts. Several model specific configuration settings for Simulink concerning data logging are performed by a script. The activation process typically takes less than a minute.

2) Model usage

After loading the Simulink model, a routine for updating the internal STOICAL data needs to be executed (see section VI) by the user scripts. Afterwards, the provided routines may be used in a control script for adjusting the STOICAL elements prior to simulation start (Tab. II).

<table>
<thead>
<tr>
<th>Element</th>
<th>Exemplary tasks and corresponding code (subject to changes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>setFunctionName(‘labelpath’, ‘value’);</td>
</tr>
<tr>
<td>Structure</td>
<td>Set all structures to default variant</td>
</tr>
<tr>
<td></td>
<td>setActiveVariant(‘#Fault’, ’3 phase fault’);</td>
</tr>
<tr>
<td></td>
<td>setActiveVariant(‘#PLL’, ‘SRF#DEF#’);</td>
</tr>
<tr>
<td></td>
<td>Setting all lines identically by regular expressions</td>
</tr>
<tr>
<td></td>
<td>setActiveVariant(‘#Line .’, ’3 phase model’);</td>
</tr>
<tr>
<td>Parameter</td>
<td>Set all parameters to default values</td>
</tr>
<tr>
<td></td>
<td>setParameterValue(‘#Line #r1’, ’0.02’);</td>
</tr>
<tr>
<td></td>
<td>Set a line parameter</td>
</tr>
<tr>
<td></td>
<td>setParameterValue(‘#IIDG 1v_pcc’, true);</td>
</tr>
<tr>
<td>Signal</td>
<td>Activate Logging of a signal</td>
</tr>
<tr>
<td></td>
<td>setLogStatus(‘#IIDG 1v_pcc’, true);</td>
</tr>
</tbody>
</table>

Routines provided for minimizing accelerator mode recompiles are discussed in the documentation. Further routines are used after simulation for either

- accessing selected signal of the last run for direct analysis / data reduction etc. or
- forwarding all / selected signals to data storage.

3) Analysis of the results

Stored signals are retrieved employing their labelpath according to Tab. III.

<table>
<thead>
<tr>
<th>Element</th>
<th>Exemplary tasks and corresponding code (subject to changes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal</td>
<td>Retrieve single signal from database</td>
</tr>
<tr>
<td></td>
<td>readSignalFromDb(‘#IIDG #1_grid’, [..]);</td>
</tr>
<tr>
<td></td>
<td>Retrieve #SIGi_grid signal of both IIDG from database</td>
</tr>
<tr>
<td></td>
<td>readSignalFromDb(‘#1_grid’, [..]);</td>
</tr>
</tbody>
</table>

The toolbox features an automatic joining of identically
named signals residing in antivalent variants of the same vari-
ant subsystem (e.g. “#SIG#v_pcc” defined in the IIDG struc-
ture variants “dq0” and “abc”). Example scripts are provided
that show best practice approaches for big signal data vol-
umes.

B. Complex model usage example

A complex example in terms of depth of model hierarchy,
high number of STOICAL structures, parameters and signals
is described in this section in order to demonstrate the
toolbox’ potentials. The system represents the coupling of two
three-phase four wire droop controlled inverters (DIVS) via
coupling lines to a common bus (PCL) in an intentionally-
landed low voltage distribution system (Fig. 7).

Figure 7. Single-line diagram of example islanded distribution system with
two droop-controlled inverters.

The use case of the model is the investigation of system
states and protection system behaviour during a grid fault. The
influence of diversely designed control structures and param-
ters as well as grid parameters shall be investigated. The in-
verter control is therefore represented in detail whilst the
modulation and high frequency switching actions are repre-
sented with an average modelling approach. Further details are
given in [5]. Fig. 8 gives an incomplete graph of the overall
model hierarchy and of the droop controlled inverters.

Details on the model complexity are given in Tab. IV.

<table>
<thead>
<tr>
<th>Field</th>
<th>Element</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulink</td>
<td>Simulink blocks / lines</td>
<td>&gt;3900 / &gt;5500</td>
</tr>
<tr>
<td></td>
<td>Simulink subsystems</td>
<td>&gt;680</td>
</tr>
<tr>
<td></td>
<td>Simulink variant subsystems (VS)</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>min. / typical / max. variants per VS</td>
<td>1 / 2 / 8</td>
</tr>
<tr>
<td>STOICAL</td>
<td>Model hierarchy units</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>max. hierarchy unit levels</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Structures / parameters / signals</td>
<td>39 / 106 / 37</td>
</tr>
<tr>
<td></td>
<td>max. levels of nested structures</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Signals after automatic joining</td>
<td>24</td>
</tr>
</tbody>
</table>

As the model is continuously developing, STOICAL in-
creases the efficiency of scripting the control mechanisms
handling the current 39 structures, 106 parameters and 37 sig-
nals.

V. DISCUSSION

A. Scope

The STOICAL toolbox is designed for research activities
and supports the user

- in efficiently performing variations of Simulink model
  parameters and model structure,
- in comparing the influence of solver and toolbox config-
 urations and
- in adequately storing and accessing the signals to log
  big simulation data volumes.

The STOICAL toolbox does not inherently include func-
tionality for

- the design of experiments but rather helps in applying
  such designs to a Simulink model and
- the variability management as often required for pro-
  duction oriented processes [3].

B. Typical use-cases and benefits

Typical circumstances creating benefits from using
STOICAL are:

- automation of executing and analysing a large number
  of experiments
- complex model hierarchies with nested structural var-
  iants or parameters on multiple hierarchy levels
• models with many similar submodels which shall be flexibly varied uniformly, group-wise or individually
• collaboration on such models by multiple researchers with different research tasks

In those cases STOICAL increases efficiency by:
• letting the user quickly define parameters, structural variants and signals to log by means of labels in Simulink block names or connection lines
• offering a meaningful syntax for addressing those elements including regular expression support
• offering unified processes and interfaces for performing variations and accessing results by scripts
• minimizing accelerator mode recompiles on structure and parameter variations

STOICAL increases productivity by letting the user focus on the research tasks rather than:
• choosing suitable Matlab/Simulink mechanisms and adapting those to changes by new releases
• designing a database structure for storing the results appropriately
• defining and adapting control variables manually in scripts and models after model changes
• joining logged signals with identical meaning defined in antivalent model structure variants manually
• routing parameters or signals through the hierarchy

Using the toolbox thereby reduces error sources typically involved in the latter tasks which are often hard to identify.

C. Restrictions

Due to the Simulink technologies used by the toolbox (Tab. I), applying the same or similar mechanisms manually disables STOICAL functionality for the affected model parts. Appliance of procedures directly manipulating mask entries (e.g. tuning functionality of PID controller) may remove STOICAL functionality from that block.

Simulink lines that are “buses” or “physical signals” may not directly be used as STOICAL signals. The documentation discusses workarounds.

Label declarations inside Library Blocks with active link or Referenced Models are not visible to STOICAL. Disabling the library link enables using STOICAL. Approaches for Referenced Models are currently investigated for future releases.

On each model load the toolbox needs to parse the model structure for the momentary Simulink session’s handles of registered elements. Parsing takes up to several seconds but needs to be performed only once for batch processing.

Parallelization is not yet directly supported by STOICAL but is currently under investigation for future releases. An overlaying process may still be used to enable usage in parallelized use cases.

D. Requirements

STOICAL is fully implemented as Matlab-scripts and therefore does not require external programs or development environments. Matlab/ Simulink and all toolboxes used by the user model itself are required.

When using a database for signal storage, the routines provided require the “Database Toolbox” and a BLOB-capable JDBC-Driver for creating a database connection. STOICAL has been extensively tested with a local SQLite3 database. The database system is adaptable by exchanging the drivers used. JDBC-drivers are not distributed with the toolbox.

Several provided scripts for optionally visualising the STOICAL element declarations in their hierarchical context make use of the “Bioinformatics Toolbox”.

The usage of STOICAL requires its naming convention to be applicable in the subject model. The used label defining markers (e.g. “#”) may be altered, but the concept of labelling itself is essential.

The additional memory requirements due to the toolbox have not been measured but are experienced to be small in comparison to the requirements of model simulation. The Simulink model file size increases by several 10-100 kB due to STOICAL data storage in the model workspace. The toolbox itself has a volume of less than 5 MB.

VI. Conclusion

The developed toolbox enables efficient execution and analysis of Simulink power system model parameter and structure variations. A complex model use case example demonstrates the toolbox potentials. The approach is generic and may be applied to other areas of research. Further work is necessary in order to overcome some current restrictions and to further extend the possible usage scenarios.

DISSEMINATION

The authors intend to publish the STOICAL source code and documentation as free and open source software [6].

REFERENCES