A Multi Level Approach for Aircraft Electrical Systems Design

† Martin R. Kuhn † Martin Otter ‡ Loïc Raulin

† German Aerospace Center (DLR)
Institute of Robotics and Mechatronics
Department of System dynamics and Control
82234 Wessling, Germany

‡ Airbus France
Electrics System
EDYNE department
31300 Toulouse, France

Abstract

This paper describes the needs, ideas, implementation and application of a multi-level concept used for aircraft electrical systems design. The goal is to easily switch between three model levels in a complex system model, in order to arrive at dedicated models for the needed simulation tasks: a simple and super fast model for energy consumption design, a detailed model for fast network stability analysis and a very detailed model for network quality assessment. Special care was spent on the modeling assumptions and a suitable library concept fitting to the needs. For simplified unitary testing and configuration management of the multi-level models a concept was developed. The approach is demonstrated with an aviation equipment use case. The usage of the models for stability and quality studies is sketched.

Keywords: Multi-level modeling, Electrical Network, Aircraft, DC-DC buck converter, stability analysis, quality analysis

1 Introduction to multi-level modeling

For industrial design, evaluation and certification process of energy distribution networks, often several models exist which represent different modeling accuracies of the same system. It is not always useful to take the most detailed one, as it may not improve the overall accuracy but will slow down the simulation drastically. Depending on the desired evaluation of an electrical network (power consumption, network stability, network quality), system simulations with models of different levels of accuracy are much better suited. The simulation time of the simplest, architectural level of a model is usually 2 to 3 orders of magnitude faster than the most complicated, behavioral level of the model.

The following model levels are taken into account:

- Level 1: Architectural level
  Steady-state power consumption. Usually, algebraic equations describing the energy balance between ports without dynamic response.
  Typical use: power budget.

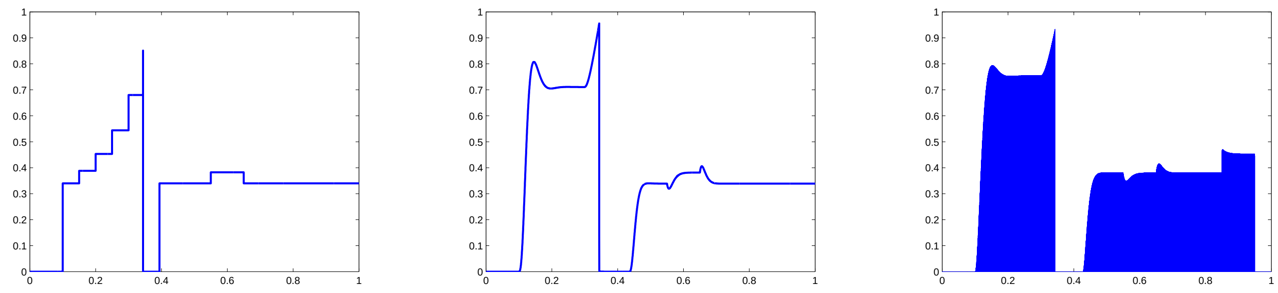
- Level 2: Functional level
  Steady-state power consumption and mean-value transient behavior (e.g. inrush current, consumption dynamics with regard to input voltage transients). Switching is not included.
  Typical use: network logic studies, network stability studies.

- Level 3: Behavioral level
  Representing actual wave forms including switching and HF injection behavior.
  Typical use: network power quality studies.

Note: This nomenclature may be used differently in the literature.

In Figure 1 on page 2 the tree-level concept is illustrated at hand of simulations of a DC/DC buck converter. The architectural layer input current shows large simulation steps neglecting the detailed effects which can be seen at the behavioral layer model simulation. The functional layer model covers the waveform of the detailed model without switching effects.

In order to improve the simulation process for design and validation of the Electrical System, there was



a) architectural layer *b) functional layer* *c) behavioral layer*

Figure 1: *Illustration of the 3 layer concept at hand of simulations of a DC/DC buck converter*

identified a need for methods to simplify unitary testing and configuration management of suppliers' models in order to enhance the coherency of electrical behavior of the various modeling levels of one single equipment.

2 Modeling assumptions

While the architectural layer is defined to contain only static energy balance equations and the behavioral models are close to the hardware level, the definition of the functional models is not as straight forward. These models have to be applicable for linearization for stability investigations with methods of control theory for linear time invariant systems. For DC systems, those have to be deduced from the complex models by averaging the switching models on a time interval and order reduction techniques.

For AC systems, the time variant characteristics of the sinusoidal alternating currents prevents a time invariant steady state condition and hence linearization. Therefore it is proposed to use an equivalent representation which expresses a rotating electric machine in a static rotor fixed system (also called Park transformation)[1]. Transforming the system equations to a net-frequency fixed system results e.g., in "i(t) = i (ω_{net}) + Δi(t)". The model uses Δi(t) as state and not i(t). In stationary operation with constant load and constant speed, Δi(t) = 0 and i(t) = i (ω). This time invariant system also results in much faster simulation. The transformation output is a representation in d, q and zero system. The zero system may be skipped for this application since it is mainly relevant for asymmetric loads not treated by functional models. Park transformation is the standard way of modeling generators and motors and used for control but is also applicable to 3phase-line impedances. Asymmetric loads are allowed but will not result in a steady state condition for this transformation. For an aeroplane multi-phase power transmission system the following components are of importance:

- Power source: The generators supply sinusoidal voltage at net frequency. Calculation of voltage and current is usually performed in a rotor-fixed system (synchronous machine)

- Loads: For symmetrical resistive, inductive and capacitive loads as well as synchronous motors current/voltage relations are fixed to net frequency.

- Other loads: Switches, diodes, unsymmetrical loads, time variant loads do not have simple steady-state dependencies upon net frequency. But: Calculation of the generator demands the use of transformations anyway. So, incorporating other components to the dq0-system will not necessarily essentially speed up the simulation but will not break it either.

Limitations for level-triggered switching devices may be circumvented by averaged models. Especially for self commutating rectifiers there is a need for calculating the mean output voltage/current, averaged on a commutation interval. Theory on electric components, the transformation and averaged models can be found for instance in [1].

[1]Some results were inspired by the EU Project Realsim (Real-time Simulation for Design of Multi-Physics Systems). It resulted in the open source library "SPOT" of H. J. Wiesmann of ABB Switzerland, http://www.modelica.org/libraries/spot

3 Library concept

3.1 Comparison of methods

Following the demands from chapter 1, methods for multi-level modeling were investigated which could replace the current models (behavioral, functional, architectural) by one single multi-level model. This multi-level model shall integrate the separate layers to be activated independently from each other. Therefore a methodology is investigated to join models of different abstraction levels within a single "container" model and supply tools for the automatic selection of the desired level. Three alternatives were identified to meet the demands:

Alternative 1: Every level is implemented as a separate sub-model. A "generic" model (or template model) is utilized in a user model via the Modelica language construct “replaceable”. Via a selection box the "generic" model can be defined to be one of the level models. The connectors of the different levels are either identical or, if this is not possible, they are also "replaceable". Details may be found in the Modelica tutorials. Typical problems are:

- It is not possible to select the level by an expression, i.e. it is not possible to select the model based on the setting of the global default. Instead, at every component the model level has to be manually changed to the required one.
- Whenever the selection is changed (e.g. from architectural to behavioral and then back to architectural), all parameter definitions from the previous selection are lost.
- All levels need the same connectors or the connectors must be replaceable, which makes the design complicated.

Alternative 2: All layers are described in the same model. Via flags different parts of the model are activated. In the simplest case, there is just an if-clause:

```
parameter Integer level(min=1, max=3) = 1 "Model
level";
...
equation
 if (level == 1) then
  // equations for architectural model
 elseif (level == 2) then
  // equations for functional model
 else
  // equations for behavioral model
 end if;
```

The simulation program selects at compile time the corresponding if-branch, if the branch is determined by a parameter expression (i.e., in the generated code, only the equations of the selected branch is present). Advantages:

- The level can be set by an Integer expression. This allows, e.g., to refer to a global setting of the level.
- The parameters of all 3 levels are visible and can be set. They remain present, even if the level is changed. This alternative is useful for base components, such as a capacitor or an inductor, that are described by equations and do not contain other model instances.

Alternative 3: This is a variant of alternative 2. Every level is a separate model. There is a "container" model that is used in a user model. The container model contains all models of the different levels in form of conditional models. Via a flag, the desired model level is activated and the connect statements to the deactivated submodels are removed automatically. Advantages:

- Every model level can be built and tested independently from the other levels. Only the connection interfaces need to be the same.
- When switching between levels, the parameter settings of the other levels remain.
- Parameters that are common to all levels can be defined in the container object and can then be propagated to the level models (via Modelica modifications)

Disadvantages:

- In the plot window there is an unnecessary hierarchy, e.g., motor.level2.flange_a.w (on the other hand, it becomes very clear which level is contained in the model).
- The connection lines from the level models to the interfaces are redundant information (e.g. not present in alternative 1)

Due to severe disadvantages with implementing this so called “conditional declaration” with existing language constructs, an initiative was started in the Modelica Association to improve this situation. As a result, in the Modelica language version 2.2 from Feb. 2005, "conditional declarations" have been introduced into the Modelica language. This language construct has been supported in Dymola since March 2005 (Dymola version 5.3c).

3.2 Implementation

Due to their advantages, it was decided to use alternative 2 for basic components, such as capacitors and inductors and to use alternative 3 for all other multi-level components (using the new Modelica feature of conditional declarations). Also test equipment, such as load resistances as function of time, may depend on the accuracy level as well as the connectors. In this section all the details are explained including library structuring.

The basic approach is to use multi-level equipment components in a system model and select in a global menu the default accuracy level. At every instance of an equipment model it is possible to define whether the default accuracy level shall be used (= default behavior) or another one via the "level" parameter. Possible options are

"global option":	Use globally defined level
"level 1":	Use architectural model level
"level 2":	Use functional model level
"level 3":	Use behavioral model level

. "outer" references the "global_options" component in the enclosing environment. Equations and/or declarations of a model are activated and deactivated depending on parameter "actualLevel" or the derived binary flags level1active, level2active or level3active. Since "actualLevel" is a parameter expression (= an expression depending only on literals, constants and parameters), Dymola evaluates conditions of if-clauses that depend on "actualLevel" at compile time and therefore selects the corresponding if-branch also at compile time. This means that any change of "actualLevel" requires re-compilation of the system model.

In order to simplify the multi-level model development, some partial models and classes can be reused. Models can inherit the multi-level components like the "outer" parameter "level" to define the desired modeling level via "extends *partialmodel*" . This is demonstrated hereafter with the multi-level model "issue1":

```
model issue1 "my model with several levels"
 extends mylib.interfaces.partial_3_levels;
 ...
 equation //for base components:
  if level1active then
   ...;
  else
   ...;
  end if;
 public //for hierarchical multi-level models:
  componentsIssue1.architecturalModel ModelLevel1
if level1active;
  componentsIssue1.functionalModel ModelLevel2 if
level2active;
  componentsIssue1.behaviouralModel ModelLevel3 if
level3active;
  ...
end issue1;
```

which extends partial_3_levels:

```
partial model partial_3_levels
 "Parent class that should be included via extend
for a multi-level model with 3 levels"
 import Choice = mylib.types.level_choice;
 parameter Choice.temp level="global option" ;
 "Model level to use (global setting,
architectural/functional/behavioral level)"
 protected
 outer mylib.components.global_options
global_options;
 parameter String actualLevel = if
(level == Choice.global_options) then
global_options.defaultLevel else level;
 parameter Boolean level1active = (actualLevel ==
Choice.level1);
 parameter Boolean level2active = (actualLevel ==
Choice.level2);
 parameter Boolean level3active = (actualLevel ==
Choice.level3);
 parameter mylib.components.global_options
global_options_temp( defaultLevel=level);
end partial_3_levels;
```

The possible choices are defined in level_choice:

```
package level_choice
 constant String global_options="global option";
 constant String level1="level 1";
 constant String level2="level 2";
 constant String level3="level 3";
 type temp
  extends String;
  annotation (choices(
   choice="global option" "use global option
setting",
   choice="level 1" "level 1 (architectural
level)",
   choice="level 2" "level 2 (functional level)",
   choice="level 3" "level 3 (behavioral
level)"));
 end temp;
end level_choice;
```

With the future implementation of Modelica enumerations in simulation environments the need for complex

naming of the choices will become obsolete.
The “global_options” object which defines the global setting has to be dragged to the highest hierarchy level and set to "inner". This defines a "global" structure that is accessible by all components on the same or a lower hierarchical level. To generate the “inner” object automatically when dragging (“inner mylib.global_options global_options; “), the following declaration is used (this property is defined via an annotation in the model):

```
model global_options
"Global options settings"
annotation (defaultComponentName="global_options",
defaultComponentPrefixes="inner",
missingInnerMessage="A \"global_options\"
component was introduced with default options.",
...
);
parameter mylib.types.level_choice_default.temp
defaultLevel= "level 1" "Default model level
(architectural/functional/behavioral level)";
end global_options;
```

Almost identical to level_choice, the global_options object supports choices for the three levels but of course no choice for global_option itself:

```
package level_choice_default
 constant String level1="level 1";
 constant String level2="level 2";
 constant String level3="level 3";
 type temp
  extends String;
  annotation (choices(
   choice="level 1" "level 1 (architectural
level)",
   choice="level 2" "level 2 (functional level)",
   choice="level 3" "level 3 (behavioral
level)"));
  end temp;
end level_choice_default;
```

As explained earlier, the multi-level container object for the alternative 3 has to be the superset of the connectors of the single models. To avoid unnecessary variables, the concept of “expandable connectors” can be employed (e.g. expandable connector positive_plug_expandable "Positive expandable electric plug"). The content of this Modelica connector is defined by the sum of connected variables. With this, non identical connectors of the level-dependent models may be connected to the container object connector but only the data of the level selected are present after compilation. E.g. for the transformed/not transformed three phase system, dq transformed components demand one extra variable in the connector: the rotor angle. On the other hand just the dq system uses 2 current/voltage connectors while the non transformed abc system uses three of them.

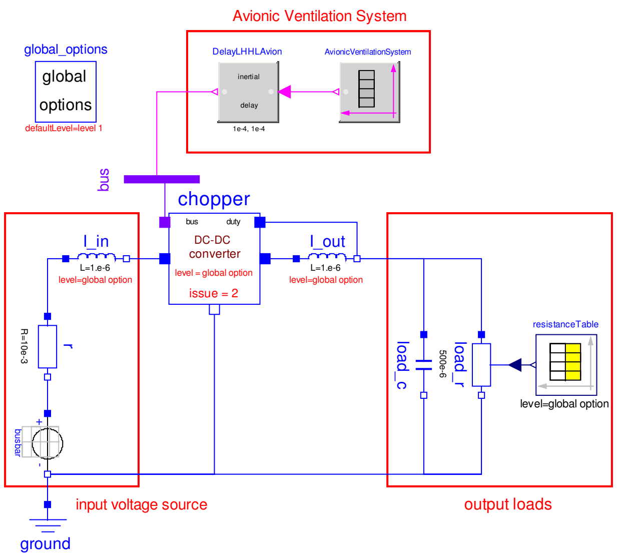


Figure 2: Test circuit for chopper use case (for all 3 levels)

4 Demonstration

In the following chapter the library concept shall be demonstrated with a generic chopper model (electrical non-isolated DC/DC buck converter). The converter is used for transformation of the 270 Volts DC supply to a 28 VDC load network. It was designed to include the most important obstacles, e.g. different levels have different interfaces, components, and load profiles. The behavioral model contains nonlinear switching semiconductors. Only with this the ripple detection for high frequency noise is applicable. The functional model is averaged with the output voltage as the duty ratio times the input voltage. The architectural model lacks of any dynamics. The converter is modeled with an energy balance. A test model for the chopper can be seen in figure 2.
Base components that are solely described by equations are implemented with if-clauses that depend on parameter “actualLevel”. For example, the inductor is defined by:

```
if level1active then v = 0;
else // level 2 or 3
 L*der(i) = v;
end if;
```

This means, for level 1 the dynamics equation is removed. In the icon of the inductor, the instance name (here: “inductor”), the inductance (here: “1.2e-6) and the value of parameter “level” (here: global option) are displayed.
Other components are implemented with container objects and conditional declarations as sketched in alter-

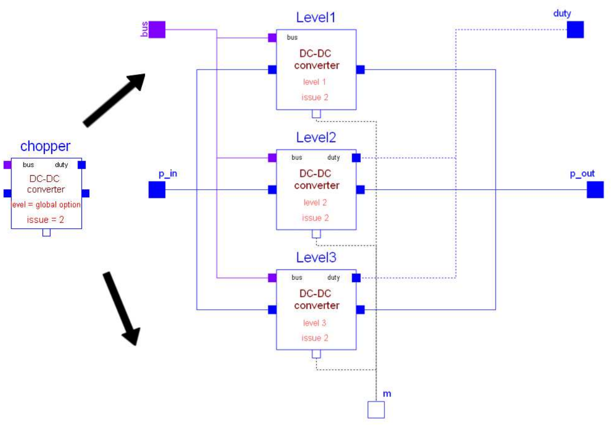


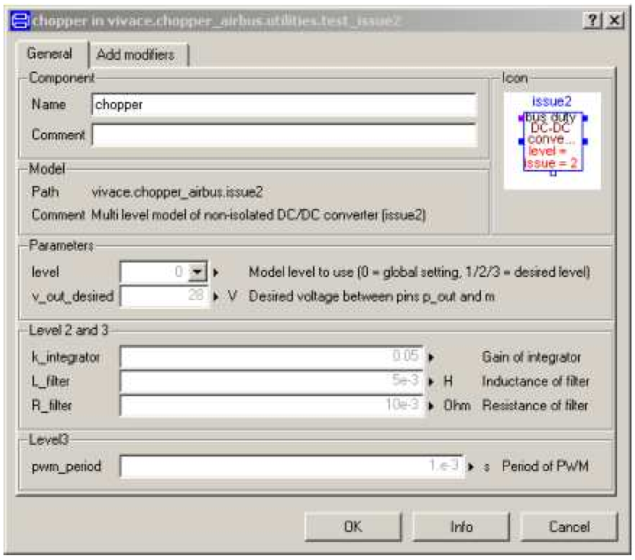
Figure 3: Multi-level chopper component with conditional declaration

native 3 above. So, the structure of the chopper model can be seen from figure 3.

In the left part of the figure, the icon of the chopper is shown. Besides the model instance name, the value of parameter "level" is displayed in the icon. In the right part of the figure, the container object is present. It contains models of every level. All models are connected to the connectors (p_in, p_out, m, duty, bus) defined in the icon and are defined by "conditional declarations":

```
model issue2
 extends mylib.utilities.interfaces.partial_3_levels
 parameter Modelica.SIunits.Voltage v_out_desired
= 28;
 ...
 mylib.chopper.componentsIssue1.level1 Level1(
v_out_desired = v_out_desired) if level1active;
 mylib.chopper.componentsIssue2.level2 Level2
( v_out_desired = v_out_desired, k_integrator
= k_integrator, L_filter = L_filter, R_filter =
R_filter)   if level2active;
 ...
 equation
  connect(Level1.p_out, p_out);
  ...
end issue2;
```

For example, component "Level1" is an instance of "mylib.chopper.comonentsIssue1.level1" and is only present if "level1active=true". If a component is used in a connection, such as "connect(level1.p_out, p_out)", this connect statement is automatically removed if one or both of the models referenced in the connect(..) statement are deactivated. Therefore, it is no longer necessary to manually include an if-clause around such connect(..) statement as in previous versions of Modelica. Note that the interface of level 1 does not include the "duty" pin but for the multi-level container object the pin is skipped if the level is level


chopper in vivace.chopper_airbus.utilities.test_issue2

General | Add modifiers

Component
Name: chopper
Comment:

Icon: issue2

Model
Path: vivace.chopper_airbus.issue2
Comment: Multi level model of non-isolated DC/DC converter (issue2)

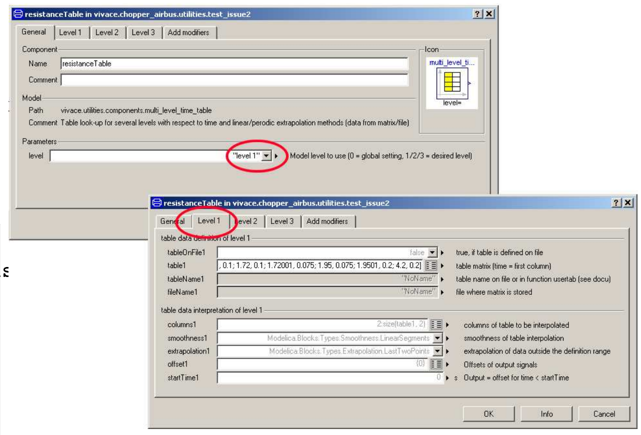
Parameters
level	0		Model level to use (0 = global setting, 1/2/3 = desired level)
v_out_desired	28	V	Desired voltage between pins p_out and m

Level 2 and 3
k_integrator	0.05		Gain of integrator
L_filter	5e-3	H	Inductance of filter
R_filter	10e-3	Ohm	Resistance of filter

Level3
pwm_period	1e-3	s	Period of PWM

OK | Info | Cancel

Figure 4: Menu of multi-level chopper component


resistanceTable in vivace.chopper_airbus.utilities.test_issue2

General | Level 1 | Level 2 | Level 3 | Add modifiers

Component
Name: resistanceTable
Comment:

Icon: multi_level_ti... level=

Model
Path: vivace.utilities.components.multi_level_time_table
Comment: Table look-up for several levels with respect to time and linear/perodic extrapolation methods (data from matrix/file)

Parameters
level: "level 1" Model level to use (0 = global setting, 1/2/3 = desired level)

resistanceTable in vivace.chopper_airbus.utilities.test_issue2

General | Level 1 | Level 2 | Level 3 | Add modifiers

table data definition of level 1
tableOnFile1	false		true, if table is defined on file
table1	, 0.1; 1.72, 0.1; 1.72001, 0.075; 1.95, 0.075; 1.9501, 0.2; 4.2, 0.2]		table matrix (time = first column)
tableName1	"NoName"		table name on file or in function usertab (see docu)
fileName1	"NoName"		file where matrix is stored

table data interpretation of level 1
columns1	2:size(table1, 2)		columns of table to be interpolated
smoothness1	Modelica.Blocks.Types.Smoothness.LinearSegments		smoothness of table interpolation
extrapolation1	Modelica.Blocks.Types.Extrapolation.LastTwoPoints		extrapolation of data outside the definition range
offset1	{0}		Offsets of output signals
startTime1	0	s	Output = offset for time < startTime

OK | Info | Cancel

Figure 5: Menu of time_table component

1. It is convenient for the user if the parameters of all levels are defined as parameters of the container object and are propagated to the corresponding models. For example, when clicking on the icon of the chopper example above, the menu shown in figure 4 is opened.

When using model level 1, only parameter "v_out_desired" is actually utilized. For model level 3, all parameters in this menu are taken into account. If the parameters of the levels are different, it is useful to display them in different "tabs". For example, in the menu of component "mylib.utilities.components.multi_level_time_table" shown in figure 5, different tables can be defined for the various levels (the purpose is, e.g., to have different load resistances for the various levels).

5 Application

Without going into detail, this chapter is a short outlook on the usage of the models for stability and quality studies.
A regulated buck converter is a typical critical component in a power system. Due to the negative resistance at low frequencies the regulated buck converter could be unstable in combination with the input filter. Therefore it is necessary to investigate the stability of the whole electric network both at small signal level for steady state conditions and large signal level for transients, impacts and network reconfiguration. The library concept proposed above is a good tool for its usage since validity of functional models can first be demonstrated by comparison of simulation results with the behavioral models. The Modelica functional models can be used for stability studies with methods for linear time invariant systems. For eigenvalue based methods, including modal analysis and eigenvalue sensitivity, the eigenvalues have to be calculated numerically by the simulation program. μ analysis is a powerful method of robust control for stability investigations of parametric varying systems. The Modelica functional models can be directly applied for μ analysis after extraction of the symbolic code and using it in Maple and Matlab. Compared with other methods for small signal stability, e.g. Middlebrook criterion and Modal Analysis, the μ sensitivity approach gives a much more global and direct result for the influence of all components on stability. For details on the methods, see [2].
In contrast to these approaches, for industrial use stability of a system often is defined as the ability of a system to keep a certain system variable within desired limits given by industrial standards. These are combined criteria of network stability, power quality and performance. This makes them difficult to proof with methods of linear control theory. Therefore a simulation based approach often is the only possibility to proof “industrial” stability and also large signal stability including failure protection devices. Instead of random or gridded parameter variation on the varying environment and system parameters, an other method is to search for the most critical parameter combination directly. The basic idea is to use an optimizer to find the criterion from the standards which is most critical and make it worst by changing the uncertain parameters in the possible range. In case the criterion is violated, stability/quality/performance can be shown to be not guaranteed. On the other hand, the tolerable design range for parameters could be investigated as the bounds leading to standard violations.
An overview on the methods is shown in figure 6.

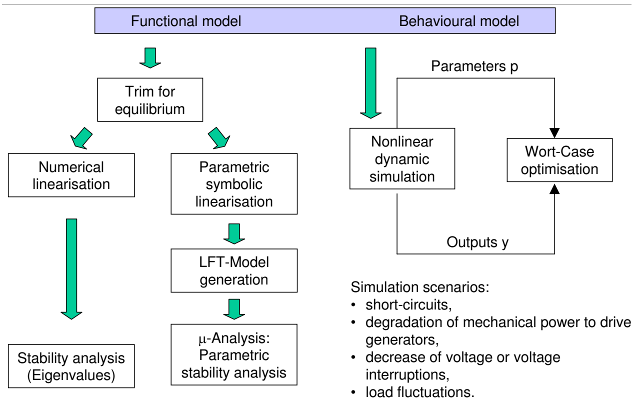

Functional model
Behavioural model
Trim for equilibrium
Numerical linearisation
Parametric symbolic linearisation
LFT-Model generation
Stability analysis (Eigenvalues)
μ-Analysis: Parametric stability analysis
Parameters p
Nonlinear dynamic simulation
Wort-Case optimisation
Outputs y
Simulation scenarios:
• short-circuits,
• degradation of mechanical power to drive generators,
• decrease of voltage or voltage interruptions,
• load fluctuations.

Figure 6: Quality and stability studies overview

6 Acknowledgment

This work was in parts supported by the EU in the "VIVACE project" (http://www.vivaceproject.com/) under contract number 502917.

7 Conclusion

In this paper a multi-level concept used for aircraft electrical systems design based on conditional declarations was shown. The three level concept was explained and modeling demands and methods were proposed, especially using park transformation for AC systems. The implementation was demonstrated with a multi level DC/DC chopper use case. The model ling concept improves modeling of large systems and allows easy comparison of different levels simulation result. An overview on typical applications of the models for stability and quality studies was given..

References

[1] Paul C. Krause, Oleg Wasynczuk, and Scott D. Sudhoff. *Analysis of electric machinery and drive systems*. Wiley-Interscience, Piscataway, New York, 2nd edition, 1998.

[2] M.R. Kuhn, Y. Ji, and D. Schröder. Stability studies of critical dc power system component for more electric aircraft using mu sensitivity. In *Proceedings of the 15th Mediterranean Conference on Control and Automation*, Athens, 2007.