Model Based Specifications in Aircraft Systems Design

Martin R. Kuhn[1] Martin Otter[1] Tim Giese[2]

[1]Institute of System Dynamics and Control, German Aerospace Center (DLR e.V.), Germany,
{martin.kuhn,martin.otter}@dlr.de

[2]Airbus operations GmbH, Germany, tim.giese@airbus.com

Abstract

This application paper describes the concept and needs on model based specifications in order to specify the basic behavior of aircraft systems and methods to check the requirements. It is demonstrated how it can be implemented by recent Modelica based libraries, especially with the new Modelica_Requirements library. Two new FFT-based requirement blocks are proposed to allow full coverage of the specification.

Keywords: executable specification, requirements, aircraft system design, FFT-based requirements.

1 Introduction

Executable specifications are computer algorithms written in an appropriate specification language with the purpose of demonstrating and verifying the compliance of the input-output behaviour of the model subject to the model specifications. In aircraft design, executable specifications demonstrating and verifying the behaviour of models can be seen as an important tool to make the co-work between airframers and suppliers quicker and more efficient, as they allow frequent testing and early validation of subsystems and systems interaction.

Similarly, requirement modelling allows the specification and testing of demands on signals which are generated by a system or the model of a system. Together, executable specifications and requirement models enable a well-defined specification of a system. Both methods allow testing against the hardware or software implementation. They strongly benefit from methods for monitoring and cross-checking.

While the traditional aircraft design process is based on document based specifications only, a model supported design process based on executable specifications and requirement models is thought to improve the process in terms of quality and time (Becker and Giese, 2011). In contrast to the traditional, more software oriented usage of executable specifications, here they were used in a more general way also for specification of physical models and behavior. In the publication the concept was evaluated with MathWorks based tools, but specification models may include physical models built with Modelica. In order to have a one-tool solution which allows better coupling of the physical models to requirement blocks, alternatives to this approach with Modelica based methods were investigated in the "CleanSky, Systems for green operation" project (Cleansky, 2015). Associated tools were developed in parallel in the CleanSky subproject ModelSSA by Dassault Systèmes, supervised by DLR-SR and in the ITEA2 "MODRIO" project with several partners[1]. This paper reviews the concept of executable specifications for aircraft systems[2] where the executable specifications are seen as a bigger package of specifications, test cases, demonstrators and monitoring functions. The implementation is solely based on Modelica.[3]

2 Review of model based design process

For aircraft systems design, the current design process is a document-based development. The behaviour of the system to be developed is defined by textual requirements, pseudo code, tables, block diagrams, logic diagrams and mathematical expressions. General demands applicable to several (sub-)systems are generalized in industrial standards, for example MIL-STD-704F (MIL704F, 2004) or airframer specific standards, for example the AirBus Directives (ABD). Document-based development has severe disadvantages: There is the danger of misunderstandings and misinterpretations of functional requirements since they are written in natural language which could result in incorrect system behaviour. Furthermore, the specified system behaviour cannot be simulated. Therefore, contradicting requirements can hardly be recognized before realization of the system. Also for multi-system functions and interfaces the validation is missing and therefore the mutual influences between systems may not be treated correctly in the early design cycles. This results in late detection of design errors when integrating the systems together. In addition, in case of requirements on signals and requirements on systems interacting with plants, the signal processing and plant test models may be implemented differently

[1] MODRIO: https://itea3.org/project/modrio.html

[2] In this paper aircraft systems (e.g. a drive) and components of a system (e.g. controller) are both called "system" to simplify notation.

[3] Section 2-4 is based on the internal reports (Kuhn et. al., 2014; Becker et.al., 2013; Becker, 2014).

between the airframer and different suppliers. In any case there might be redundant work since the same monitoring functions need to be implemented by several suppliers or at the airframer for testing.

In contrast, by a Model Based Design Process (MBDP) the system to be developed is specified by models representing the functional and/or physical behavior. The models can be delivered with test environments and monitoring functions which are modeled by the airframer. To prevent confusions and double work, it is essential to tightly link the documentation and the model based specification. This can be achieved by automatic generation of the documentation from the model and its embedded requirements and optionally with linkage of models to requirements in databases. By this approach, misunderstandings and lack of information is avoided since the models provide a mathematically precise definition and allow interactive simulation and investigation. While the model should cover all aspects of the systems' functionality, there is no necessity to express all of it in a single model. A combination of methods as

- flow diagram notation,
- state transition notation,
- physical modelling, plus the afore-mentioned
- written requirements

can be used. This methodology was evaluated in (Becker and Giese, 2011). The efficiency of the model based approach was analyzed in (Becker et al., 2013). In (Becker, 2014) the model-based design process was tested qualitatively against the former development programs. It could be shown that the model based specification process results in reduced cost for development of control systems. Those are significant advantages from a project management perspective.

3 Elements of an executable specification model

In the following we will introduce representative requirements in the style of (Tunnat, 2011) and (MIL704F, 2004) for the Environmental Control System (ECS) and for the electrical system.

For a model-based design process, the requirements can be grouped into two different layers:

- The high level requirements.
- The functional requirements.

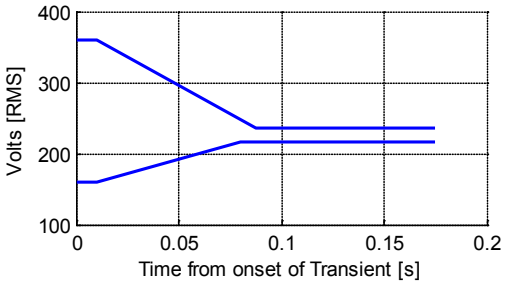
The high level requirements treat specifications on the exterior behavior of the system. The formulation is based on engineering knowledge and top level demands. The **high level requirements** include

(1) Demands on the implementation and realization. Those requirements generally are not stated by use of models. A requirement can be stated as in R1.

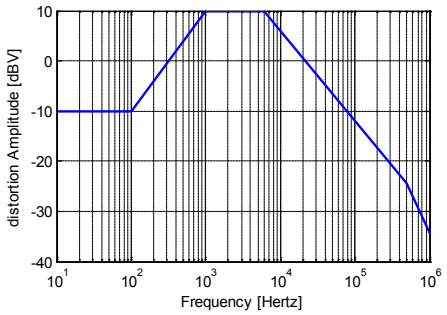
R1: The engine's total probability of failure must be smaller as 1e-9.

(2) Demands on the signal and state behavior. Such requirements are stated by requirement blocks that assess the specific requirements using observation variables from a physical model as part of the executable specification. Requirements may be specified based on industrial standards. A requirement can be stated as in R2 and R3.

R2: In normal operation mode, the envelope of the RMS value of the 400 Hz AC voltage after a voltage transient is given by the following figure and has to remain in the final limits.



R3: In normal operation mode, the distortion spectrum of the variable frequency AC voltage has to remain below the limits given by the following figure.



In contrast to the high level requirements, the **functional requirements** define the realization and logic of the system to be realized in an abstract but executable language. For example, R4 and R5:

R4: In case Ditching is not active, the OVBDV shall be in its PO position and the BUV shall be in its FO position, five seconds after Override has been activated

R5: For a two-position valve defining a valve flap the following functional behaviour shall apply:

If the system is in state "open" indicated by "full_open"=true, a commanding signal "closing" without indication "fault" shall result in state "close". In case of "fault" the system shall go into condition "open_fault" with indicator "stuck_open"=true. Reciprocal rule for state "close" with indicator "full_close", command opening and fault condition "close_fault" indicated by "stuck_close"

In these examples, the logic is functional as no details on the physical realization is given.

DOI 10.3384/ecp15118491

A fully model based design process relies on extensive modelling to express demands by functional modelling, supplied test environments, model of the physical system with its components, test it against the specification and check the result. The center in charge of the aircraft or system specification may cover all or a selection of the following tasks to express the model based specification for a component (or system) which shall be developed:

Table 1: Elements of model based specification process

1	Specification of the components functional and procedural behavior by models.
2	Simple physical model to demonstrate the desired behavior of the component and allow simulation with physical test environments.
3	Physical modelling of the testing environment.
4	Expression of requirements and modes of operation by requirement monitors.
5	Definition of interfaces to physical states, environmental states, logical states.
6	Mapping of requirements to the interfaces of the functional models.
7	Providing property monitors with built in signal operations for requirements which demand advanced processing of interface signals-
8	Providing indicators and automatic documentation of warnings and faults.
9	Make tools available for managing requirements and documentation

In the systems realization phase of the supplier and afterwards in the systems integration phase of the airframer there are additional demands for

- Systematic testing of system models.
- Clearance of requirements.

It is the task of the supplier to realize the system and harmonize the behavior of the executable specification and the developed system. For this, the model of the developed system can be embedded into the test model, being optimized and checked by the monitors.

4 Realization with tools of MathWorks

The aforementioned approach was evaluated by Airbus Germany at hand of a controller design of the ECS. The model of the controller could be best modelled by hierarchical state charts and stateless flow charts.

The modeling platform used several packages and tools of the MathWorks product family. The **physical system** of the ECS system architecture was modelled with Simulink. Alternatively, Modelica models can be imported in Simulink. For the hybrid **state space modeling** of component models, Stateflow was used (MathWorks, 2015b).

The **functional specifications** of the controller make use of Stateflow as well. The state diagrams give a detailed description on the systems behavior, including start sequence, transitional conditions and entry conditions when changing to adjacent states.

For managing of requirements, no ready to use product was found which met the demands. Thus a special requirement manager was commissioned by Airbus (toolbox developed by Silver Atena[4]).

The requirement manager summarizes and documents the requirements, tracks the requirements changes and allows some coverage analysis on requirements with predefined test scenarios. It relies on additional special properties block which are inserted to the local functional model. For this part the "Verification and Validation" toolbox (MathWorks, 2015a), the Airbus requirement manager, and Simulink's "Report Generator is used.

An example of a model based specification for an ECS controller is shown in Figure 1 formally defining Requirement 5. The pneumatic network is the physical plant which has to be stabilized by a controller to be developed. The pneumatic system acts as environment model and is realized by Simulink blocks. The preliminary model of the controller can be implemented in a very simple manner at this stage. The only aim is that the physical system can be simulated, even if the simulation results violate requirements. For example, in case a physical demonstrational model is needed, the controller could be implemented as a P controller while the supplier's realization may rely on a sophisticated model-based controller.

In addition to the preliminary controller - and more important - the functionality of the controller (R5) is defined by additional Stateflow diagrams. They are the result of a pre-design at the airframer. In the right part of Figure 1, requirements for the behavior in case of errors are defined. The system can be simulated and checked interactively by variation of the input states of the Stateflow system.

No special monitors for high level requirements were implemented.

After implementation of the functional executable specification, the formal verification of requirements can be realized with the "Design Verifier" block set from MathWorks. An example is shown in Figure 2 formally specifying requirement R4.

The blocks in the left calculate the requirement while the „statement" block is linked to a requirement checker and monitoring system. The system supports documentation and formal verification of the requirements. Other special monitors for high level requirements can be implemented with Simulink blocks or Simulink S-functions.

[4] http://www.silver-atena.de

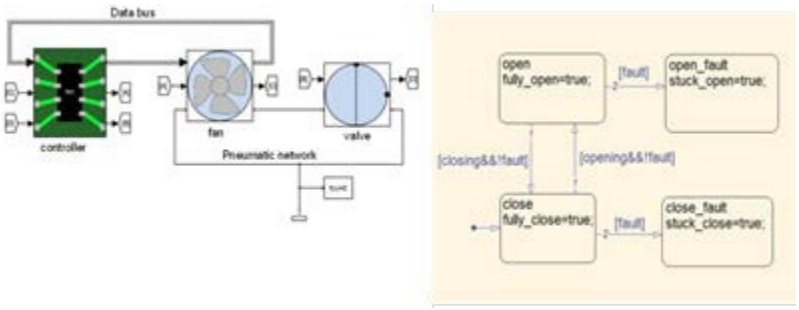


Figure 1: Model based specification at hand of an ECS example

"*In case Ditching is not active, the OVBDV shall be in its PO position and the BUV shall be in its FO position, five seconds after Override has been activated.*"

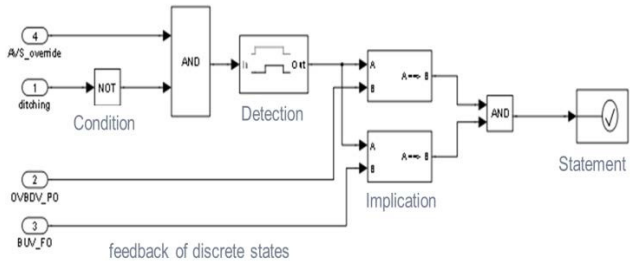


Figure 2: Formal specification of requirement R4 using the Design Verifier from MathWorks.
Figure and text from (Tunnat, 2011).

5 Realization with Modelica

This section shows how to use Modelica for modelling and checking of requirements to provide the necessary functionality of Table 1. In general, the transition from the paper based design process to model based specification, executable specifications and automated testing is mostly a matter of the development philosophy rather than of the technical realization.

Functional requirements can be most conveniently specified in Modelica by synchronous state machines (Elmqvist et al., 2012). In Figure 3 the example of Figure 1**Fehler! Verweisquelle konnte nicht gefunden werden.** is shown, implemented with a Modelica synchronous state machine.

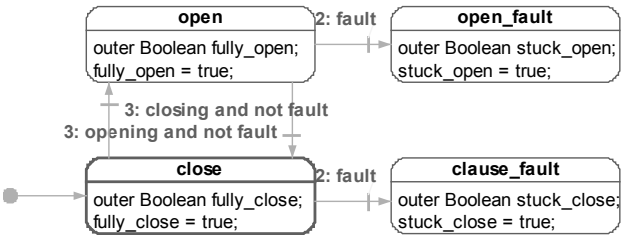


Figure 3: Modelica synchronous state machine of the example in Figure 1.

The realization and user friendliness is mostly equivalent to a Stateflow implementation in this case. However, the Modelica synchronous state machines have a more rigorous definition to avoid modelling errors. For example, there may be assignments to the same variable in different state machine "states" (such as in state "close_fault" in Figure 3. These state machine states are "mutually exclusive" and at one sample instant the code of only one of these states is executed. Furthermore, in parallel state machines, exactly one assignment to the same variable at the same sample instant is allowed. In essence, Modelica and a Modelica tool only allow one single assignment to the same variable at one sample instant, in order to always have deterministic, well-defined behaviour. On the other hand, in Stateflow several assignments to the same variable are possible.

Alternatively, one may express the system in Modelica by **behavior trees** which follow a slightly different concept. Behaviour trees can be used for modelling of logical behavior and especially mission planing. Complex missions are built up using atomic tasks. Tasks can query conditions from system states or trigger actions, e.g. by sending commands to the communication bus. For a detailed description of the Modelica library used here, see (Klöckner, 2014).[5] The main advantage of behavior trees for executable specifications is their standardized and intuitive structure to express alternative paths. Plans are very scalable and human-readable on all levels of the hierarchy. It is their benefit and drawback at the same time to be inherently memory- and loop-free. They thus execute the correct task immediately after a restart or online modification.

This is demonstrated in Figure 6 which is the Modelica behavior tree implementation of the ECS example of Figure 1. Depending on the Boolean input variables *fault* and *opening*, one of the four conditions *open*, *close_fault*, *open_fault* or *close* occurs. The logic is like this: Starting always from the top, a *selector* tries to execute one of the paths linked below, where the preference is from left to right. The "*sequence*" starts a sequence from left to right, in case the breaking condition (II-Symbol) is true. For example in case of "*not opening*", the "*selector*" cannot take the left path "*sequence*" which is blocked by "*condition*". Instead the right path to *selector2* is tried. "*sequence2*" ends in the action "*open_fault*" in case "*faultyO*" is un-blocked by *fault = true*, and otherwise in "*close*".

[5] Different types of "behavior trees" in a Modelica context have also been used in (Myers, 2010).

DOI 10.3384/ecp15118491

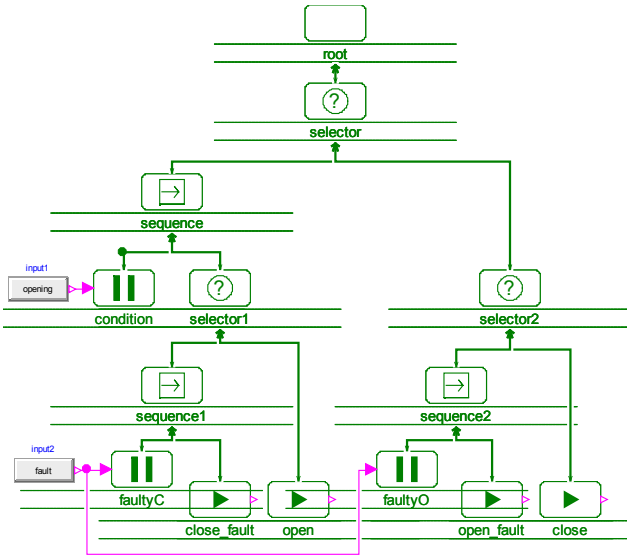


Figure 4: Modelica behavior tree of ECS example in Figure 1.

Another important demand of model based specification are physical demonstrator models of the system and modelling of test and environment models (Table 1, demand 2 and 3). Obviously, Modelica is very well suited for this part, due to the many available physical modeling libraries, that are much more intuitive to use than with only graphical input/output block diagrams.

The expression of high level requirements can be formulated in principle by any type of mathematical operation which results in an expression for requirement fulfilled/not fulfilled (or not yet evaluated). For example in (Kuhn, 2011) Modelica requirement models have been designed for band constraint signals or frequency domain constraints. The textual output of the requirement checking was based on Dymola proprietary scripting and was missing systematic output and documentation concepts.

In parallel to JTI activities, the European ITEA projects EUROSYSLIB, OPENPROD, and their successor MODRIO also identified a strong need for requirements modelling. Their approach resulted in the new Modelica_Requirements library (Otter et al., 2015). One essential advantage of this library is that it uses two- and **three-valued logic** to specify requirements. It is then possible to distinguish whether a requirement is *satisfied*, *violated*, or *not tested* during a simulation. It could be demonstrated in the JTI project, that the requirements library fulfills many needs for formulation of executable specifications of the electrical and the ECS system. In particular, for the examples in this paper, the LogicalBlocks, the TimeLocators and the ChecksInSlidingWindow have been used.

Requirement R2 could be implemented with the Modelica_Requirements library with several BandDuration blocks. A more convenient approach is sketched in section 5.2 by using a newly designed and implemented "Funnel" block.

Frequency domain requirements, such as needed for Requirement R3, cannot be defined with the current Modelica_Requirements library. Therefore, new requirement blocks have been developed based on the Fast Fourier Transformation (FFT), see section 5.3.

An implementation of requirement R4 with the Modelica_Requirements library is shown in Figure 5. The requirement block "*requirement_AVS_override*" displays the textual version of the requirement in its icon and collects the status of all requirement blocks during one simulation run. By this example it is also o

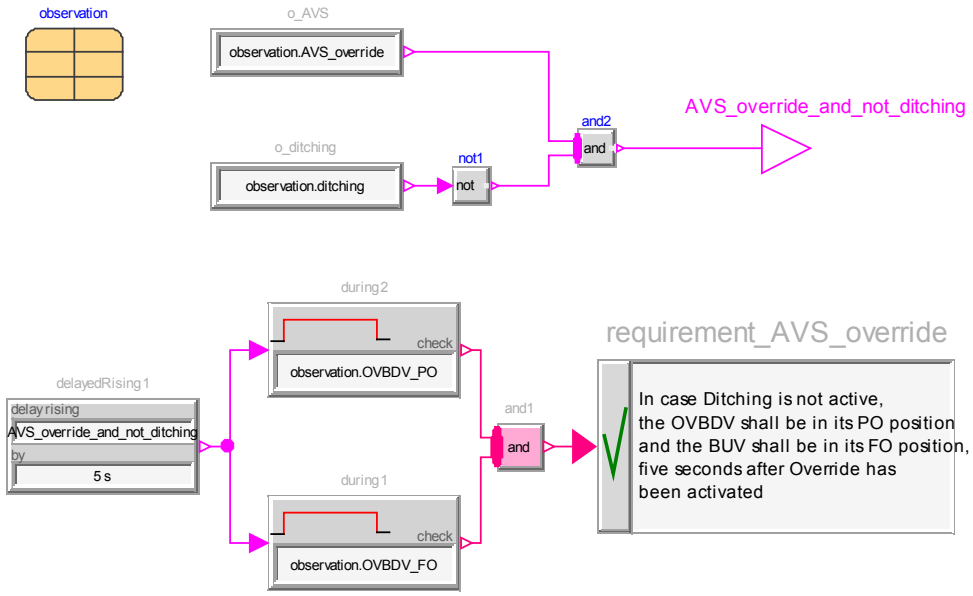


Figure 5: Formal specification of requirement R4 with the Modelica_Requirements library.

demonstrated how to bind requirements to the physical model: The general idea is to define observation variables in Modelica records, as needed from a physical system model ("observation" in Figure 5). Via newly developed Modelica language elements the actual values of the observation variables can be inquired conveniently from the physicale system model (Elmqvist et al., 2015). Figure 6 shows the final status with the requirement model (lower left) and the system model (upper part). The system model may be the executable specification or the supplier's model in the verification phase.

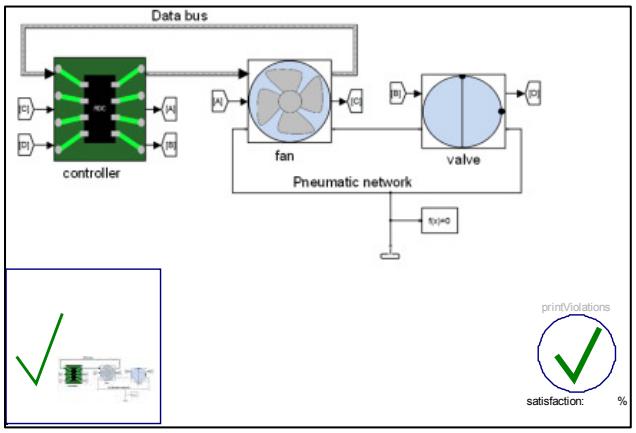


Figure 6: Binding and assessment of ECS requirements.

The requirement model is linked to the system model by the following instantiation of the requirement:

```
Requirements.AVSRequirements Req1(
  observationName="controller",
  observation= Bindings.AVSRequirementFromController(
                                              controller))
```

Bindings.AVSRequirementFromController is a function to map the variables from the controller to the requirement record. observationName defines the name of the target of the requirement. This is needed for automatic documentation.

At the end of the simulation, the following log is displayed:

```
--- 100 % of the requirements are satisfied ---
Requirements satisfied (1 of 1):
Controller(Req1.requirement_AVS_override):
In case Ditching is not active, the OVBDV shall
be in its PO position and the BUV shall be in
its FO position, five seconds after Override has
been activated
```

The current development stage allows to check in every simulation run whether the defined requirements are satisfied or violated (or are not tested).

The binding concept is flexible enough to bind requirements to all instances of a class using the experimental component iterators. For example in case of multiple controllers, an iteration would map a requirement to each controller.

For organization of functional expressions, there exists no requirement manager similar to the Simulink solution for Modelica yet. The Simulink tool summarizes and documents the requirements, tracks the requirements changes and allows coverage analysis on requirements with predefined test scenarios. In the MODRIO project, further developments are planned in this direction.

5.1 Demonstration: Specification of hardware

As last example, the concept of a model based design process relying on a model based specification shall be demonstrated at hand of a realistic example of the design of a generator. Alternatively to written specifications, the airframer may deliver a model based specification. The test model is shown in Figure 7 on the right side (supply of linear resistive three phase load and nonlinear rectified load to investigate the harmonics in the AC line).

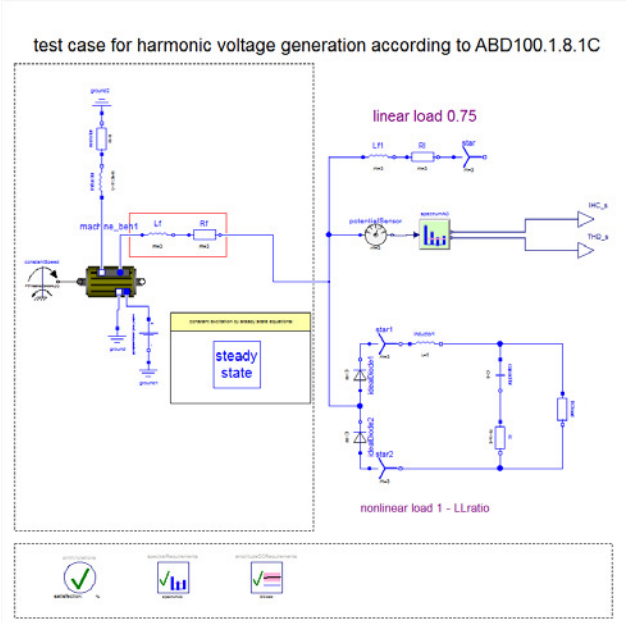


Figure 7: Demonstration of testing environment with requirement models and signal monitors.

The availability of test models allows easy and uniform implementation for all suppliers. Special operations on signals needed for the requirements checking might be also given as models. Here, the green block embeds an FFT based requirements blocks, an alternative realization (Kuhn, 2011) to the FFT block of section 5.3.

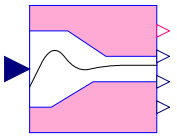
The requirements are stated by Modelica blocks. In this case two requirement blocks are associated to the model in the lower right corner which check the requirements for the operating area of the DC voltage and frequency content of the AC line voltage.

A primitive generator model might be supplied by the airframer. This is replaced by the supplier by a much more detailed model (dashed box in the left). By this test environment, the generator model can be tested and also optimized in relation to the requirements.

DOI 10.3384/ecp15118491

The model in the left lower corner triggers the summary log of the requirement blocks. The output together with the documentation of the test model and the system model (generator) are valuable parts of a proper industrial model delivery.

5.2 Transient Limits monitor



The "Funnel" block, displayed in the figure to the right, allows checking of transient time limits in funnel style. The upper and lower limits are defined via a table versus time. The initial start of the time varying limits is triggered by an initial overshoot of the limits. The initial limits are defined by the final band. This funnel type limit may be retriggered if one full period of the funnel style limitation has gone by. The output y indicates the satisfaction of the criterion. Further outputs are a scaled distance to the limits and the time varying upper and lower limits.

5.3 FFT-based frequency property monitor

Frequency based criteria are typical for industrial standards of electrical systems but are not yet supported in the Modelica_Requirements library. For example, MIL-STD-704F (MIL704F, 2004) defines a maximum distortion in the spectrum of the 270Volts DC system.

Based on the implementation and practical experience with the FFT monitoring block of (Kuhn, 2011), two FFT blocks were newly designed and implemented. An example of the user's view of the new FFT block *WithinAbsoluteFFTdomain* is shown in Figure 8. An alternative block with limits for total harmonic distortion (THD) is shown in Figure 9.

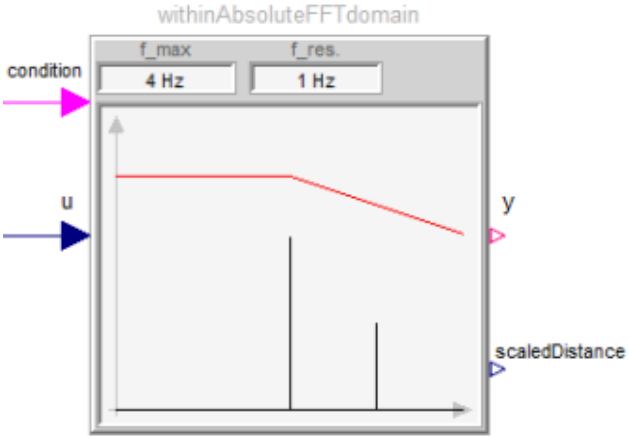


Figure 8: Example of WithinAbsoluteFFTdomain block for the inputs: $\boldsymbol{u = 2 + 3 \cdot \sin 2\pi f_1 t + 1.5 \cdot \sin 2\pi f_2 t}$ ($\boldsymbol{f_1 = 2}$ **Hz**, $\boldsymbol{f_2 = 3}$ **Hz**) and `condition = true`.

The user interfaces were designed to allow parameterization with a minimum of information and display the amplitudes over the frequencies in the icon.

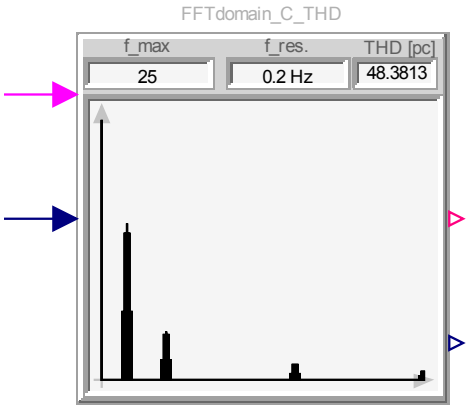


Figure 9: Example of WithinAbsoluteFFTdomain_THD block with: $\boldsymbol{u = 5 + 3 \cdot \sin 2\pi f_1 + 1.5 \cdot \mathrm{pulse}(2\pi f_2)}$ ($\boldsymbol{f_1 = 2}$ **Hz**, $\boldsymbol{f_2 = 5}$ **Hz**, "pulse" is the rectangular pulse function at frequency $\boldsymbol{f_2}$) and `condition = true`.

In the icon of *WithinAbsoluteFFTdomain*, the two scalar parameters of this block are displayed, `f_max` – the maximum frequency of interest for the user, and `f_res` – the resolution of the frequency axis (so the increment of the frequency axis). Typically, the user is interested in a maximum frequency `f_max` that is an integer multiple of some base frequency (e.g. 50 Hz base for power distribution networks). The frequency resolution should be selected in such a way, that the spectral lines of particular interest are an integer multiple of the resolution (in order to get the most accurate result). In the example `f_max = 4 Hz` and `f_res = 1 Hz`, so 5 frequency values are shown in the icon (0, 1, 2, 3, 4 Hz). For numerical reasons, in practice the resolution should be chosen high enough to distinguish well between adjacent peaks in the spectrum.

The constraints for the frequency amplitudes are defined via a polygon based on a tabular parameter input. Typically, there are two kinds of parameterizations: Definition via absolute values for the constraints and definition in relation to the magnitude at a certain frequency. For relative definition, the user is requested for the respective base frequency. In case this frequency is not an integer multiple of the frequency resolution, the frequency closest to it is taken. With parameter *searchInterval* a search interval around this base frequency is defined, where the maximum peak in this region is taken as real base frequency. For example for the 50 Hz net frequency of the European power grid, the frequency may vary by ±0.2Hz in regular operation mode. After initialization, the limits are displayed as red polygons in the icon

Whenever the Boolean input `condition` has a rising edge, the Real input signal `u` is periodically sampled with a sample rate automatically computed from `f_max` and `f_res` and stored in a buffer. Once "sufficient" values are stored in the buffer (for details, see below), an FFT is computed, displayed in the icon as bar plot and stored on file. Additionally, the distance

to the amplitude boundary is computed. If at least one amplitude is above the boundary, output y = Violated. If all amplitudes are below the boundary, y = Satisfied, and if the FFT has not yet been computed, y = Undecided. In the example of Figure 8, y = Satisfied.

In case a falling edge of u occurs before sufficient sample values are monitored or the simulation run is terminated, then the FFT spectrum is approximated via the partly-filled buffer with zeros for other values (called "zero-padding" technique).

Standard tools/functions for FFT provide a different, *user-unfriendly* parameterization. The mapping of the parameterization of the *WithinAbsoluteFFTdomain* block to the underlying standard FFT parameterization is non-trivial and is shortly sketched:

In order that the *amplitudes* are computed by the FFT with sufficient precision, the FFT computation needs to be performed for a much larger frequency as of interest for the user. In the block a fixed factor of 10 is used. So, if $f_{max} = 4$ Hz, then the FFT computation uses internally a maximum frequency $f_{max,FFT} \geq 40$ Hz. The basic formulae for an FFT computation of real numbers with even number of sample points are summarized in equation (1):

$$\begin{aligned} f_s &= \frac{n_s - 1}{T_s}, \\ f &= \left[0, \frac{f_s}{n_s}, \frac{2f_s}{n_s}, \ldots, \frac{f_s}{2}\right], \\ \Delta u_r &= u(t_r) - u_{DC}, \\ n_f &= \frac{n_s}{2} + 1 \\ u_{FFT,k}(f_k) &= \frac{1}{n_f} \sum_{r=0}^{n_f - 1} \Delta u_r \, e^{-i2\pi k \frac{r}{n_f}} \end{aligned} \tag{1}$$

where

- T_s is the sample period.
- n_s is the number of sample points
- f_s is the sample frequency $\left(f_{max,FFT} = \frac{f_s}{2}\right)$
- $\frac{f_s}{n_s}$ is the frequency resolution (`f_res`).
- n_f is the number of frequency points
- u_{DC} is the arithmetic mean of the signal
- Δu_r is the difference of the input signal with respect to the arithmetic mean u_{DC}.
- u_{FFT} is a complex number as function of a (real) frequency $f_k, k \in [1..n_s]$ and represents the FFT.

In order to be efficient, the original FFT algorithm by Cooley and Tukey (Cooley, 1965) requires that the number of sample points is an integer multiple of 2: $n_s = 2^i, i = 1,2, \ldots$ Newer algorithms allow more prime numbers. The implemented blocks use the public domain C-code KISS FFT (Borgerding, 2003). This mixed-radix FFT code requires that the number of sample points must be an integer multiple of 2, 3 and 5: $n_s = 2^i 3^j 5^k$. For real signals, n_s must be additionally an *even* number.

The maximum frequency $10 \cdot f_{max}$ is now enlarged so that the number of sample points n_s fulfills the above restrictions. The sample period T_s is determined, so that the frequency resolution f_s/n_s has the required value. These computations are performed with the following Modelica code:

```
// Compute best ns according to 10*f_max and f_resolution
ns :=2*integer(ceil(10*f_max/f_res));

// Make ns even
ns :=if mod(ns, 2) == 0 then ns else ns + 1;

// Find smallest ns that is even + expressed as 2^i*3^j*5^k
while true loop
  ns1 :=ns;
  while mod(ns1,2) == 0 loop ns1 :=div(ns1, 2);end while;
  while mod(ns1,3) == 0 loop ns1 :=div(ns1, 3);end while;
  while mod(ns1,5) == 0 loop ns1 :=div(ns1, 5);end while;
  if ns1 <= 1 then  break; end if;
  ns :=ns + 2;  // enlarge ns, but keep it even
end while;

// Compute other FFT variables
f_max_FFT = f_resolution*div(ns, 2);
Ts        = 1/(2*f_max_FFT) "Sample period";
T         = (ns - 1)*Ts     "Simulation time";
```

To understand the numbers above beforehand, utility function showNumberOfFFTpoints(..) is provided that computes them. For example calling the function as

```
showNumberOfFFTpoints(f_max=2000, f_resolution=27);
```

results in the following output:

```
Desired:
  f_max          = 2000 Hz
  f_resolution   = 27 Hz

Calculated:
  Maximum frequency used  = 20250 Hz
  Number of sample points = 1500 (=2^2*3^1*5^3)
  Sample period           = 2.46914e-005 s
  Simulation time         = 0.0370123 s
```

Note, that

$$\begin{aligned} f_{max,FFT} &= (n_f - 1) \cdot f_{resolution} \\ &= \frac{n_s}{2} \cdot f_{resolution} \\ &= \frac{1500}{2} \cdot 27 \text{ Hz} \\ &= 20250 \text{ Hz} \end{aligned}$$

In the "advanced" tab access is given to parameters less often used:

- *SearchInterval* (search interval around base frequency)
- *TerminateAfterFFT* (When true, the simulation is terminated after evaluation of the FFT)
- Parameterization of the "Window" type

In case the sampled interval does not match a multiple length of the occurring waves, the spectrum would

DOI 10.3384/ecp15118491

suffer from this "discontinuity" of non-matching levels at start and end point since the FFT assumes periodic signals. This can be circumvented by multiplication of the time series by a filter of the same length, called "window function". If this window function exhibits a shape with zero at start and end and some maximum in the middle, this discontinuity can be attenuated. By choice of a proper window function, erroneous high frequency signals will be diminished and the signal power at frequencies not precisely matched in the FFT output spectrum is smeared to the adjacent spectral points (called bins). For details see (Heinzel, 2002). The influence of windowing is demonstrated in Figure 10 and Figure 11. A sinusoidal signal of amplitude 1.5 and frequency 3.4 Hz is not matched by the FFT's output resolution of 1 Hz. Figure 10 shows a peak at 3 Hz with an amplitude of 1.2, some amplitudes in the adjacent bins and content for all higher frequencies.

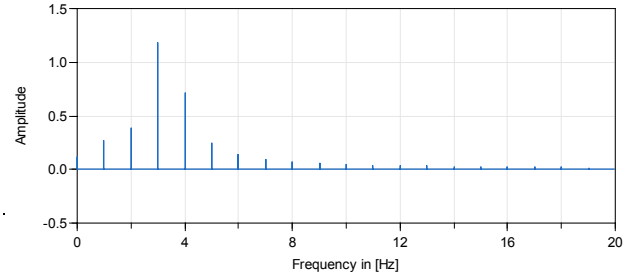


Figure 10: $\boldsymbol{u = 1.5 \cdot \sin(2\pi \cdot 3.4 \cdot t)}$ and 1 Hz resolution.

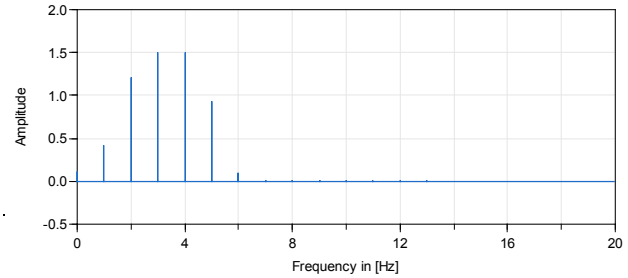


Figure 11: $\boldsymbol{u = 1.5 \cdot \sin(2\pi \cdot 3.4 \cdot t)}$, 1 Hz resolution and flat top window.

In contrast, Figure 11 is the FFT output of the signal which was windowed by the "Minimum sidelobe 3-term-at top window SFT3M" (Heinzel 2002) of length n_s with the window

$$w_i = 0.28235 - 0.52105 \cdot cos\left(1 \cdot 2 \cdot \pi \cdot \frac{i}{n_s - 1}\right) + 0.19659 \cdot cos\left(2 \cdot 2 \cdot \pi \cdot \frac{i}{n_s - 1}\right), \quad i = 0..n_s - 1 \tag{2}$$

One can see from the plot, both frequencies 3 Hz and 4 Hz show the amplitude of the original signal of 3.4 Hz. Also the next bins show a higher (erroneous) content while there are only low amplitudes at higher frequencies. As a consequence it is recommended to use windowing only in case where discrete peaks in the spectrum are expected, which may not be matched well by the resolution, the output resolution is low and the information about the correct amplitude is essential.

In addition to the *WithinAbsoluteFFTdomain* block, the *WithinAbsoluteFFTdomain_THD*, calculates the Total Harmonic Distortion (THD) from the FFT output. THD is a measure for the amplitudes of harmonics in relation to the amplitude of the base frequency, where M is defined by $f_{base} \cdot M \leq f_{max,FFT}$:

$$THD = \sqrt{\sum_{k=2}^{M} A[k \cdot f_{base}]^2} \, / A[f_{base}] \tag{3}$$

The THD criteria should only be evaluated for periodic steady state conditions. Periodic steady state is typically only occurring after an initial transient phase of the simulation. Instead of using an arbitrary settling time, the block offers the following feature: The THD can be evaluated cyclically at quite low numeric cost and is assumed to converge to a steady state value at periodic steady state condition. The *WithinAbsoluteFFTdomain_THD* block offers the option to evaluate the THD cycle every `update` % of the base harmonic until the difference between two successive THD evaluations is below `changerate`. At this point the criterion is calculated.

In Figure 12 some benchmarks for different kinds of data storage of the n_s FFT points is given:

- *SamplingAndModelicaBuffer* (= blue line) buffers the data at every sampling interval $1/f_s$ in a Modelica array. Due to Modelica's single assignment rule, all values of this array need to get a value at every sample instant. If a value is not changed at the current sample instant, the value from the previous sample instant is copied (so at every sample instant $n_s - 1$ values are copied).
- *SamplingAndBuffer* (= red line) invokes a C function at every sample instant that stores the actual value of the input signal into an internal C array.
- *NoEventConly* (= green line) does not use sampling but a C function stores the input value at every model evaluation into an internal C array. The values in this array are interpolated and internally sampled before the FFT is computed. For older Dymola versions this was beneficial since the simulation restart after a sample instant was "expensive" for a stiff solver. For newer Dymola versions this is not the case if the sampled system does not influence the integrator (which is the case here).

As can be seen from the figure *SamplingAndModelica-Buffer* is the slowest. *NoEventConly* is a bit faster as *SamplingAndBuffer*. In other benchmarks, *SamplingAndBuffer* is the fastest approach. Due to these benchmarks, in the two blocks the *SamplingAndBuffer* approach is used for data storage.

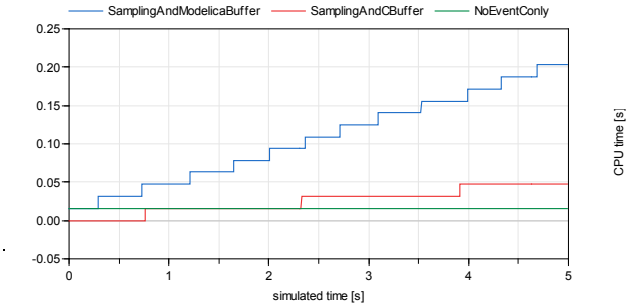


Figure 12: Comparison of CPU time [s] for three types of data storage for the FFT points.

6 Summary

In this paper the concept of model based specification and associated tools for aircraft systems was discussed. While previous work on this subject is based on MathWorks toolboxes it could be shown that Modelica could be used instead. Especially the new Modelica_Requirements library adds important extensions to express high level requirements and bind requirements to the system model under study. In combination with the FFT based requirement blocks of this paper, the full range of typical aircraft requirements for electrical systems can be formally defined. For automated documentation additional tools and scripts tailored to the need of the airframer or supplier is needed.

7 Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2016) for the Clean Sky Joint Technology Initiative under grant agreement no. CSJU-GAM-SGO-2008-001.

References

Becker C., and Giese T. (2011). Application of model based functional specification methods to environmental control systems engineering. *SAE Paper : Aerotech Congress & Exhibition*.

Becker C. et.al. (2013). Efficiency of model based methodologies in air systems engineering. *AST Workshop on Aircraft System Technologies*.

Becker C. (2014). Modellbasierter Entwurf von Flugzeugklimasystemen: Herausforderungen und Nutzen funktionaler Systemspezifikationen. *Technical report, Airbus Germany, EYVVC*.

Borgerding M. (2003). Kiss fft. URL: http://sourceforge.net/projects/kissfft/.

CleanSky (2014). Deliverable D2.1.4: Simulation and Design Platform Report. Revision b. *Technical report, Cleansky SGO*.

CleanSky project (2015). Systems for green operation (sgo). URL: http://www.cleansky.eu.

Cooley, James W.; Tukey, John W. (1965). "An algorithm for the machine calculation of complex Fourier series". *Math. Comput.* **19**: 297–301. doi:10.2307/2003354

Elmqvist H., Gaucher F., Mattsson S.E., and Dupont F (2012). State Machines in Modelica. *Proceedings of the 9th International Modelica Conference*, Munich, Germany, Sept. 3-5. Download: http://www.ep.liu.se/ecp/076/003/ecp12076003.pdf

Elmqvist H., Olsson H., and Otter M. (2015). Constructs for Meta Properties Modeling in Modelica. *Accepted for Modelica'2015 conference*.

G. Heinzel, A. Rüdiger and R. Schilling (2002). Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new at-top windows. URL: http://www.rssd.esa.int/SP/LISAPATHFINDER/docs/Data_Analysis/GH_FFT.pdf

Klöckner A. (2014). The Modelica BehaviorTrees Library: Mission Planning in Continuous-Time for Unmanned Aircraft. *Proceedings of the 10th International Modelica Conference*, pp. 727 –736, Lund , Sweden, March 10 -12. DOI: 10.3384/ ECP 14096727. Download: http://www.ep.liu.se/ecp/096/076/ecp14096076.pdf

Kuhn M.R. (2011). Advanced generator design using pareto-optimization. *Power Electronics and Drive Systems (PEDS)*, 2011 IEEE Ninth International Conference on, pp. 1061 –1067, Dec. DOI: 10.1109/PEDS.2011.6147391.

Kuhn M.R., and Ji Y. (2014). Modelica for large scale aircraft electrical network V&V. *Proceedings of the 10th International Modelica Conference*, pp. 747-756. DOI 10.3384/ECP14096747. Download: http://www.ep.liu.se/ecp/096/078/ecp14096078.pdf

MathWorks (2015a). Simulink Toolbox: Verification and Validation. URL: http://www.mathworks.com/products/simverification/.

MathWorks (2015b). Stateflow. URL http://www.mathworks.com/products/stateflow/.

MIL704F (2004). MIL-STD-704F: Aircraft electric power characteristic. Download: http://everyspec.com/MIL-STD/MIL-STD-0700-0799/MIL-STD-704F_1083/

Myers T., Geoff Dromey R. and Fritzson P. (2010). Comodeling: From Requirements to an Integrated Software/Hardware Model. *IEEE Computer*, vol.44, no. 4, pp. 62-70, April 2011

Otter M., Thuy N., Bouskela D., Buffoni L., Elmqvist H., Fritzson P., Garro A., Jardin A., Olsson H., Payelleville M., Schamai W., Thomas E., Tundis A. (2015). Formal Modeling and Automatic Verification of Requirements. *Accepted for Modelica'2015 conference*.

Thuy N. (2014). D2.1.1 – Modelica extensions for properties modelling, Part III: FOrmal Requirements Modelling LAnguage (FORM-L). *Internal report, ITEA2 MODRIO project*, Sept. 2014.

Tunnat M. (2011). Integration modellbasierter Methoden in den Entwicklungsprozess hybrider Flugzeugregel-ungssysteme am Beispiel des Ventilation-Control-System. *Master thesis, Technical University Hamburg-Harburg, Institut für Flugzeug-Kabinensysteme, supervised by C. Becker and T. Giese (Airbus)*.