Formal modeling of system properties for simulation-based verification of requirements: lessons learned

Francesco Aiello, Alfredo Garro
Department of Informatics, Modeling, Electronics and Systems Engineering (DIMES)
University of Calabria
Via P. Bucci 41C, 87036 Rende (CS), Italy

Yves Lemmens, Stefan Dutré
Siemens PLM Software
Interleuvenlaan 68
B-3001 Leuven, Belgium

***Abstract*—Requirement analysis, modeling and verification are an important part of the development process. There is a strong need for integrating these aspects into a formalized model-driven development process, together with a dedicated methodology as well as effective tool-chains. In this context, the paper presents a Modelica-based implementation of an approach for the formal modeling of system properties and the simulation-based verification of requirements. The tool-chain and the workflow adopted are described. The solution is applied to evaluate different design variants of a trailing-edge high-lift system. Two ways to feed the requirements model are explored: in an early phase, data series are used to evaluate the requirements themselves; then a co-simulation of the requirements model with the 3D-model of the system is used to evaluate and identify what design variants best meet the system requirements. Furthermore, the lessons learned from the experimentation, pros and cons, what needs to be solved about the approach, and the steps that it currently misses are discussed.**

Keywords— Formal Properties Modeling; Requirements Engineering; Model-Based Systems Engineering; Modeling and Simulation; Modelica; System Verification.

I. Introduction

To succeed in today's competitive marketplace, engineering organizations must adapt to rapid technological change and satisfy a continuous demand for new products and technologies. Innovations increase cyber-physical systems complexity. This complexity comes from the sub-systems as they are more and more heterogeneous, interconnected and interdependent.

Maintaining the compliance between the requirements and the system under consideration becomes progressively difficult and unproductive if the design process is based on documents and specifications represented in natural languages. This does not allow to identify errors and integration inconsistencies, which comes out later in the development process generating additional costs.

To address these challenges, system engineering methodologies for complex systems design make increasing use of modeling and simulation techniques. The main aims are to support functional validation of system requirements, design verification against requirements, testing, dysfunctional analyses, and verification of operational procedures. Verification is the confirmation process, through the provision of objective evidence that specified requirements have been fulfilled; its purpose is to ascertain that each level of the implementation meets its specified requirements. It is essential to utilize tools that can guarantee an objective checking of the models and of their generated values.

Models provide a single, consistent, and unambiguous representation that ensures integrity and eases system implementation and verification throughout the whole lifecycle. Virtual engineering enables the understanding of the system before it is being built; e.g. unanticipated behaviour due to unforeseen interactions can be discovered by computer simulations.

Modeling of system properties deals with formally expressing constraints and requirements that influence and determine the structure and behaviour of a system. The idea behind property modeling is that an higher model and scenario quality may be achieved when using a formal representation of system requirements. Requirements are expressed by modeling the properties that shall be fulfilled by the system. Formalizing requirements with property modelling improves the quality of the requirements themselves by removing ambiguities, omissions or inconsistencies.

It is expected that the model of the requirements may be used as an observer to conduct the verification test automatically to detect possible violations in the requirements, and that it will be possible to generate automatically test scenarios from the property models. Automating the production of test scenarios and test runs should improve significantly the test coverage and therefore the demonstration that the system operates properly.

Modelling and simulation-based verification of system requirements is an area of active research. In the ITEA MODRIO project [7], a complete approach for a typical industrial scenario was developed: first defining the requirements for a system, then performing an architectural design that shall comply with the requirements and finally evaluating and fine-tuning the architectural design with behavioural models.

Furthermore, in the MODRIO project the FOrmal Requirements Modeling Language (FORM-L) was developed to describe requirements in a formal way but close to the (textual) notation used by system designers. FORM-L was evaluated and refined on a larger benchmark example [8].

In further works, it was systematically evaluated how to map FORM-L language elements and ideas to Modelica [6], an object oriented and equation-based language for the modeling and simulation of cyber-physical systems with acausal features. These efforts finally resulted in the Modelica_Requirements library [3][9].

In this context, the paper presents a workflow to analyse, model, and verify requirements as well as how it can be implemented using Modelica based tools. The open source Modelica_Requirements library is exploited. The solution is applied to the aerospace context for the evaluation of an aircraft subsystem component and its design variants against the requirements.

The paper is organised as follow. Section II mentions the approach proposed in the MODRIO project for simulation-based verification of requirements, and gives an overview on the FORM-L language used for the properties modeling. Section III presents a Modelica-based implementation of the proposed approach and a related tool-chain, alongside the workflow adopted. In Section IV the solution is exploited for the requirements verification of a trailing-edge high-lift system. Lessons learned and future improvements are finally delineated in Section V.

II. An approach for simulation-based verification of requirements

In the context of the MODRIO project, a new architecture to automate the verification of requirements using simulation is proposed: it suggests defining requirements formally, then designing the architecture of the system, and to provide the behavioural model to evaluate the state of the design. Finally, associate requirements and architecture with behavioural models to verify the system design against the requirements. The formal model of the requirements is used as an observer of the behavioural model to detect automatically violations in the requirements. In the following subsections, each step of the approach is discussed, and further information are given to explain how it is possible to handle a requirements model.

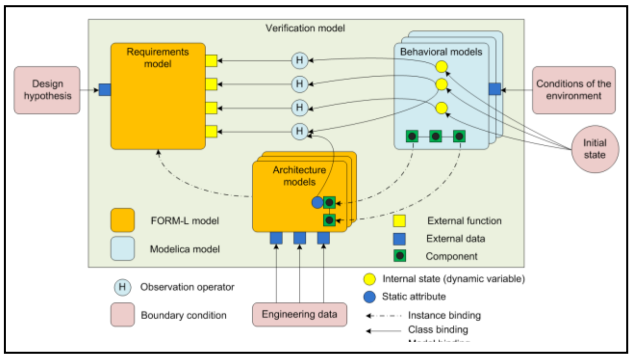


Fig. 1. Approach defined within the Modrio project for simulation-based verification of requirements.

A. Requirements model

A system property, according to [5], is an expression that specifies a condition that must hold true at given times and places. System properties can be regarded as *assumptions*, *requirements*, and *guards*. An assumption is a property that is supposed to be satisfied (e.g. that a simulation scenario assumes / ensures that is satisfied). A guard is a condition that must be satisfied for a system to be valid. Requirements are attributes, conditions or capabilities that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents [4]. System requirements are defined to ensure the proper operation of complex physical systems (such as power plants, aircraft or vehicles), but also to state functionality that satisfies customer needs. Usually they involve all the steps of the system's lifecycle.

Requirements models may be expressed using standardized graphical annotations based on the UML or SysML standards (e.g. ModelicaML). However, graphical annotations often lack the semantic rigor needed to express requirements without ambiguity. The objective of FORM-L (FOrmal Requirements Modelling Language) [8] is to combine both the semantic rigor needed for automatic processing and the language expressiveness to be understandable by operation engineers. It allows to model systems properties as assumptions, requirements, and guards.

The expression of a property in FORM-L addresses four questions: (i) WHAT is to be satisfied, (ii) WHEN in time the WHAT needs to be achieved, (iii) WHERE in the system the WHAT needs to be achieved, (iv) HOW WELL the WHAT needs to be achieved (as real-life system can and will fail).

Examples of FORM-L expressions, related to properties of a Backup Power Supply (BPS) [8], are shown in the following to describe the main constructs of the language.

R1: The BPS system must not be active when it is under maintenance.

```
required property R1 =
  during bps.state == maintenance //WHEN
  check not Active; //WHAT
```

The WHEN part is specified through *continuous time locators* (CTL) or *discrete time locators* (DTL). A CTL defines one or more time periods that have certain duration and that may overlap (see R1); whereas, a DTL defines one or more instants in time with no duration (see R2).

R2: The BPS must be deactivated when it goes under maintenance.

```
required property R2 =
  when bps.state becomes maintenance //WHEN
  check Active becomes false; // WHAT
```

In R1 and R2 the WHAT part defines a *condition-based* property to be checked. Other alternatives are *event-based* properties and *actions*. An *event-based property* specifies a constraint on the number of occurrences of an event during the time periods defined by the time locator. *Actions* can be used to specify a desired sequence of activities that is expected to be performed (see [8] for examples).

The WHERE part states the components of the system concerned by a property. This can be expressed by naming the individual components if they are well known at the time of the property model definition or using the notion of *set* of which the members will be provided later.

HOW WELL puts a limit on the probability that a desirable property is not satisfied (given the fact that real life systems are bound to have failures). As an example, in the following expression, property P8 states that within 60 seconds after the power loss by the Main Power Supply System (mps.eLoss), when not in maintenance, object *sbc* should be repowered. Requirement R8 states that the probability of not providing power to *sbc* when needed must be less than a *threshold* (x1):

```
property P8 =
  after mps.eLoss and not maintenance within 60*s
  check sbc.powered;
required property R8 =
  after P8.notTested becomes false
  check probability P8.violated < x1;
```

A property model is a set of inter-related FORM-L based property declarations and definitions that constitute a meaningful whole. Some declarations can be external and represent *inputs* to the property model. Property models can be organized into a hierarchy according to the system decomposition levels (e.g. System → Subsystems → Equipment → Components).

B. *Architectural models*

The architectural models describe the overall design of the system. They classically decompose the system into subsystems or components, express interactions between them, and place design requirements and assumptions on each of them. They also specify the overall system behaviour in terms of when and where specific actions are performed within the system.

C. *Behavioral models*

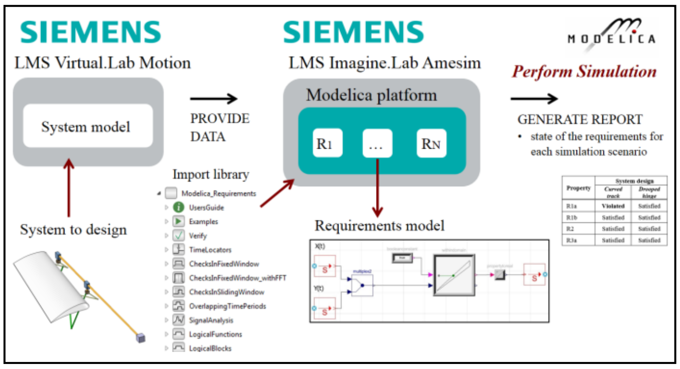
The behavioural model is built by assembling model components that contains the equations describing the dynamic behaviour of the components. The model components must be connected according to specific rules that ensure that the behavioural model is mathematically and physically well formulated.

D. *Binding*

The binding represents the possibility to establish a match between different system models (such as architectural, behavioural and property) and enabling their composition. Observation operators and bindings are used to ensure that the behavioural model may be developed independently from the requirements, and may be linked to the requirements model for simulation. There are different possibilities to provide data to the property models: (i) a mathematical model describing the physical phenomenon; (ii) data series, stored in files, coming from measurements of real experiments or simulations; (ii) co-simulation of the property and the system model. Typically, data series are initially used to evaluate the requirements themselves; then a co-simulation with the model of the system is used to automatically verify whether the requirements are satisfied.

III. A MODELICA-BASED IMPLEMENTATION OF THE PROPOSED APPROACH AND A RELATED TOOL-CHAIN

This section defines the elements of a solution (depicted in Fig. 2) that enables the formal modeling of requirements in Modelica and their subsequent simulation-based verification. The solution is applied to evaluate different design variants of a trailing-edge high-lift system using the outcomes of the requirements checking as criterion for the comparison.



Property	System design	
	Curved track	Drooped hinge
R1a	Violated	Satisfied
R1b	Satisfied	Satisfied
R2	Satisfied	Satisfied
R3a	Satisfied	Satisfied

Fig. 2. Tool chain.

A. *Modelica_Requirements library*

Modelica_Requirements is a software library (see the main packages in Fig. 2) based on the Modelica language, that implements the constructs provided by FORM-L to enable the visual modeling of system properties as well as their verification through simulation. It provides a subset of FORM-L operations but also extensions.

B. *LMS Imagine.Lab Amesim*

The tool-chain employs LMS Imagine.Lab Amesim [10] as simulation platform. Amesim is an advanced 1D modeling environment for performing simulations and analysis of multi-domain physical systems. It contains many libraries created using C, but it also supports Modelica.

The Modelica platform of Amesim refers to a collection of tools that allow to create, view, modify and compile models written in the Modelica language. Moreover, it permits to import external libraries such as *Modelica Standard Library* and *Modelica_Requirements*.

Once imported, the *Modelica_Requirements* library enables the creation of property models which are then compiled to native LMS Amesim sub-models. At that point, the main LMS Amesim platform handles the simulation and post-processing as it would with any native LMS Amesim library sub-model. The input ports of Amesim sub-models are used to provide data to the Modelica property models, while the output ports give back the results of requirements assessment for plotting.

C. LMS Virtual.Lab Motion

The simulation model of the system under investigation is implemented in LMS Virtual.Lab Motion [10], an integrated multi-body solution to model, simulate and analyse the realistic dynamic motion of any mechanical system. The co-simulation is performed between Amesim and Motion by exporting the model defined in Motion as an Amesim sub-model, to execute the equations of the various software blocks separately and exchanging data at discrete time periods.

D. Workflow

The adopted workflow can be summarized as follow: given the system requirements in natural language: (i) Model the requirements with FORM-L; (ii) Map the FORM-L representation of requirements to Modelica using the *Modelica_Requirements* library; (iii) Compile the property models to obtain Amesim sub-models with input/output ports; (iv) Bind sub-models to data series from files or co-simulation sub-models of Motion to feed the property models with data; (v) Perform simulation to analyse the outcome of requirement verification by means of plots or results files generated by the *Verify* package of the library.

Step (i) of the above sketched process is optional; however, although systems requirements can be directly modeled using the *Modelica_Requirements* library, producing a preliminary FORM-L based representation of them can be useful both to better understand the requirements themselves and to derive in a more seamless and effective way their Modelica-based representation by exploiting the constructs provided by the library.

TABLE I. MATCH BETWEEN MODELS AND TOOLS

Model	Tool
Requirements model	Modelica Platform of Amesim, Requirements library [9]
Architectural model	SysML [11]
Behavioral model	LMS Virtual.Lab Motion [10]
Binding	LMS Imagine.Lab Amesim [10] Co-simulation and Data Series

IV. A CASE STUDY IN THE AEROSPACE DOMAIN

The case study concerns a trailing-edge high-lift system of a commercial transport aircraft. In aircraft design, it is a component or a mechanism on an aircraft's wing that is deployed to increase the amount of lift produced by the wing when required.

A. Architectural model

The system architecture (see Fig. 3) consists of a flap, deployment mechanisms, actuators, gearboxes, shafts, a motor and a wing attachment structure. Four variations of the architecture were made by having common but different mechanisms realizing the deployment function. Variations are realizations of the architecture with different component models or different characteristics to change design variables. Thus, there are four design variants of the system according to the kind of deployment mechanism (see Fig. 4) employed in the architecture: (a) Drooped hinge (b) Four bars (c) Curved track (d) Hooked track.

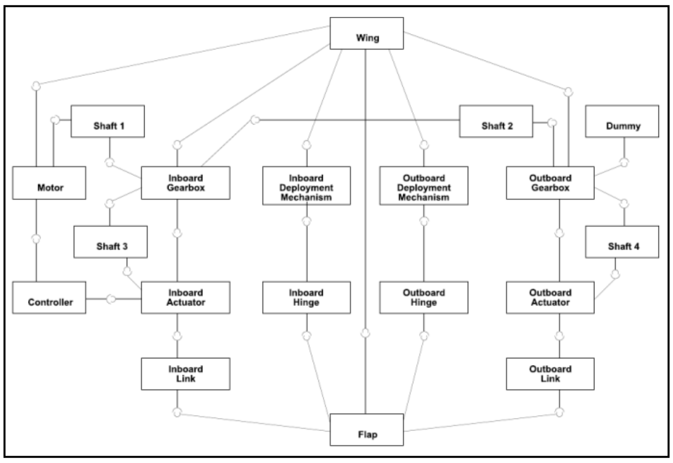


Fig. 3. Architectural model of the high-lift system.

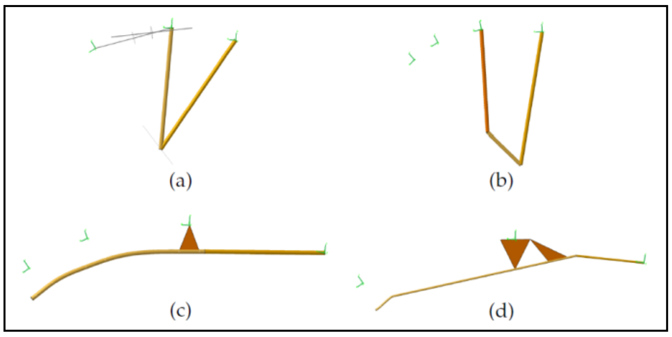


Fig. 4. CAD models of the deployment mechanisms that define the design variants of the system [2].

B. Behavioral model

The multi-body model of the high-lift system was developed in the master thesis [2]. It has different levels of detail about the components, for different purposes: (i) parameters for kinematic sizing; (ii) detailed geometry for structural sizing; (iii) finite element analysis to consider flexibility of components.

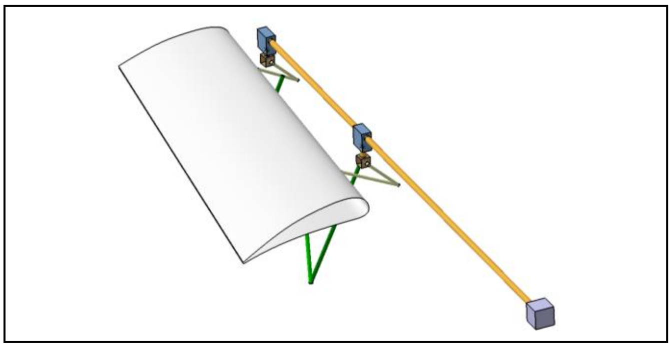


Fig. 5. Virtual.Lab Motion 3D simulation model of the high-lift system [2].

C. *Requirements Model*

The main purpose of high-lift device is to give the aircraft acceptable take-off and landing performances. It only has an aerodynamic purpose; therefore, its functional requirements are only related to aerodynamic behaviour. The top-level requirement of the high-lift subsystem is to translate and rotate the flap surface during different flight stages to obtain satisfactory performance of the aircraft (system level) [2]. It is noted that the geometrical variables involved by the requirements are measured from a reference system on the flap surface.

Functional requirements - The high lift device shall position the flap surface to satisfy aerodynamic requirements during the phase of take-off and landing (2D kinematic characteristics R1, R2, R3).

- R1a: The longitudinal position of the flap surface shall be into $[X_{min}, X_{max}]_{\textbf{take-off}}$ when the deployment angle (DA) is in $[D_{min}, D_{max}]_{\textbf{take-off}}$
- R1b: The longitudinal position of the flap surface shall be into $[X_{min}, X_{max}]_{\textbf{landing}}$ when DA is in $[D_{min}, D_{max}]_{\textbf{landing}}$
- R2: The lateral position of the flap surface shall be constrained during the longitudinal translation to avoid the contact with the wing surface and to ensure a diagonal translation on the Y-X plane.
- R3a: The vertical position of the flap surface shall be into $[Z_{min}, Z_{max}]_{\textbf{take-off}}$ when the longitudinal position is in $[X_{min}, X_{max}]_{\textbf{take-off}}$
- R3b: The vertical position of the flap surface shall be into $[Z_{min}, Z_{max}]_{\textbf{landing}}$ when the longitudinal position is in $[X_{min}, X_{max}]_{\textbf{landing}}$

Performance requirements are introduced to evaluate and to compare the design variants regarding feature of interest that should be considered during the design process (R4, R5, R6).

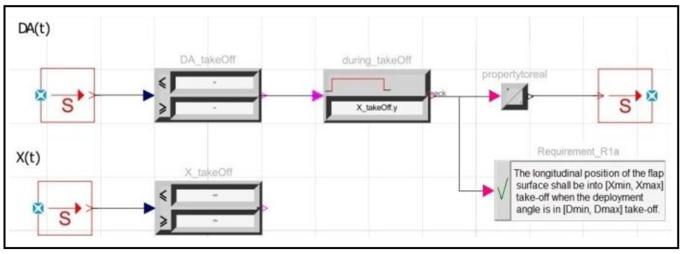
- R4: The sensitivity angle shall be constrained, to ensure a smooth variation of DA during the deployment. This variable is obtained as the first derivative of DA respect to the longitudinal position: $Y<K_{SA}$
- R5: The height of the fairing shall be constrained to limit the aerodynamic drag. It is the vertical distance between the bottom skin of the wing and the lowest point of the mechanism: $h(t) < K_{FH}$
- R6: The maximum torque employed by the motor shall be minimized to reduce the required motor size.

In the following it is shown how the requirements R1a and R5 could be modeled in FORM-L.

```
propertyModel Req_HL_device
  propertyModel R1
    external Real DA, X;
    parameter Real DA_min_Toff=14, DA_max_Toff=16;
    parameter Real X_min_Toff=300, X_max_Toff=400;
    required property R1a =
     during (DA>DA_min_Toff and DA<DA_max_Toff)
     check (X>X_min_Toff and X<X_max_Toff);
  end R1;
  propertyModel R5
    external Real h; parameter Real K_FH=0.9;
    required property R_FH=
     check (h<K_FH);
  end R5;
end Req_HL_device;
```

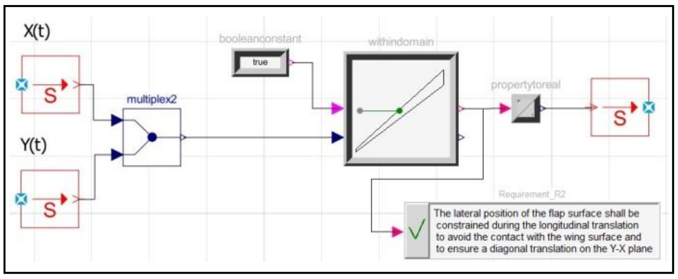
In the model of R1a (see Fig. 6) the Real variables coming from Amesim are input of the *WithinBand* blocks, which Boolean output is true if the input is within the range specified by the parameters. Thus, two signals (*condition* from DA and *check* from X) are generated and given as input to the *During* block. During a *condition* phase the output value is *Satisfied* if *check* is true, *Violated* otherwise. When *condition* is false the output is *Undecided*, suggesting that the property is not tested.

In the model of R2 (see Fig. 7) two Real variables (representing the longitudinal and the lateral position of the flap surface) are given in input to the *WithinDomain* block, which checks that the 2D-input point (x,y) is within an area defined by a closed polygon. The shape of the polygon (see Fig. 6) is defined by means of its vertices. The “Requirement_R2” block collects the status of R2 during a simulation run.



The longitudinal position of the flap surface shall be into [Xmin, Xmax] take-off when the deployment angle is in [Dmin, Dmax] take-off.

Fig. 6. Possible implementation of R1a with the Modelica_Requirements library.



The lateral position of the flap surface shall be constrained during the longitudinal translation to avoid the contact with the wing surface and to ensure a diagonal translation on the Y-X plane

Fig. 7. Possible implementation of R2 with the Modelica_Requirements library.

In the model of R5 (see Fig. 8) the Real variable (representing the height of the fairing) is given in input to the *LessThreshold* block, which checks that the input is less than a threshold defined as a parameter. The block checks the value during all the simulation run, as it is not defined a specific time locator.

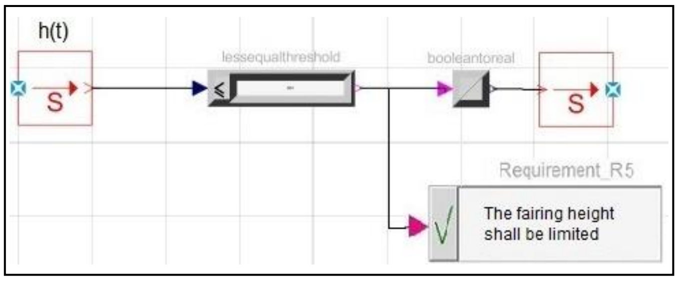


Fig. 8. Possible implementation of R5 with the Modelica_Requirements library.

D. Binding

Once the system requirements are modeled in FORM-L and implemented with Modelica sub-models, the binding with the system model can be made. Fig. 9 shows the Amesim simulation environment and the binding between system and property models by a Co-simulation with a Motion sub-model. Thus, many test cases can be performed, also allowing varying the test scenario and the parameter of the system.

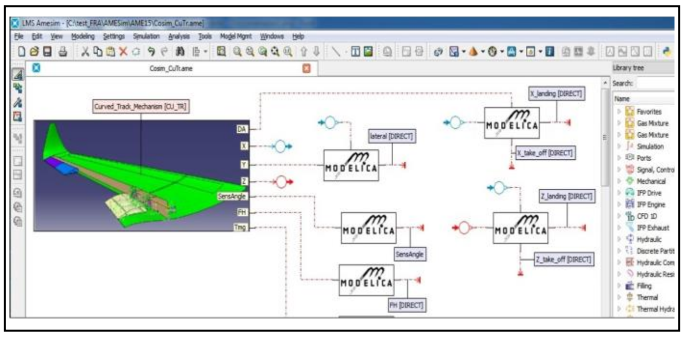


Fig. 9. Connection between system and property models in LMS Imagine.Lab Amesim by Co-simulation with a sub-system exported from LMS Virtual.Lab Motion.

E. Simulation and Results Analysis

The objective of the simulation is to evaluate the state of the requirements and, possibly, to compare the different design alternatives. In Fig. 10 a 2D representation of requirements R1a and R1b is shown, together with the trajectory for the deployment of the four design variants. The output of a simulation run for a system design shows the state of the requirements over time, for the input scenario, which is a combination of the system and requirements parameters.

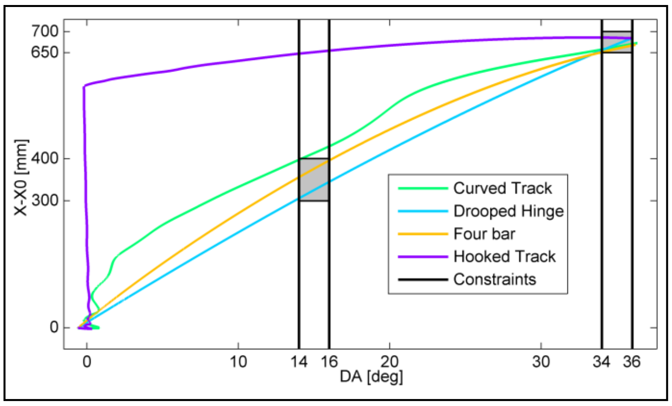


Fig. 10. Simulation of the four design variants respect to the 2D representation of the requirements R1a (take-off scenario) and R1b (landing scenario).

TABLE II resumes the results for the different design alternatives. Obviously, they depend by the chosen parameters for the constraints. By the results emerges that the Drooped hinge mechanism fulfils most of the requirements and requires the minimum motor torque to deploy the flap surface. The last row enumerates the design variants according to the requirement R6, which is about the maximum torque employed by the motor (e.g. I stand for minimum torque, IV maximum torque).

The library allows also generating textual reports of requirements assessment for each simulation run, which can be useful to build report documents when a large amount of test is performed.

TABLE II. DESIGN ALTERNATIVES COMPARISON

Property	**System design**			
	Drooped hinge	***Four bars***	***Curved track***	***Hooked track***
R1a	Satisfied	Satisfied	**Violated**	**Violated**
R1b	Satisfied	Satisfied	Satisfied	Satisfied
R2	Satisfied	Satisfied	Satisfied	**Violated**
R3a	Satisfied	**Violated**	Satisfied	Satisfied
R3b	Satisfied	Satisfied	Satisfied	**Violated**
R4	Satisfied	**Violated**	**Violated**	Satisfied
R5	**Violated**	**Violated**	**Violated**	Satisfied
R6	I	IV	II	III

V. Lessons Learned

The formal requirements specification, respect to the document-based definition, has the following main advantages in terms of requirements management: (i) a reduction of the ambiguity and an increasing in the accuracy, due to the well-defined syntax and semantic of the formal language adopted; (ii) the improvement of the efficiency of the co-work between system manufacturers and suppliers as property models provide a shared and reference representation of systems requirements that can guide testing and early validation of system and subsystem interactions.

The possibility to perform a simulation-based verification presents the following benefits: (i) it lets to understand rapidly if the current implementation is not compliant with the requirements, starting from very early stages in system design; (ii) it enables the comparison of different design alternatives and parameters settings, respect to the specifications, during the design stage. Indeed, as it allows to explore more quickly the domain of the feasible solutions, it is possible to get to the optimal one in an effective way. This improves system design optimization solutions in terms of quality and time; (iii) it allows monitoring the system behaviour against the requirements during the operation phase. What happens while the system is working in several real-life situations? How do the different scenarios affect its performance and functioning? How does it perform with an unforeseen test case? (iv) it permits to analyse the system operation in case of fault-injections and potentially implement fault-tolerant mechanisms. Given the property model and the simulation model, with co-simulation designers can consider and experiment solutions as well as verify their effectiveness before to apply them to the system; (v) in case the requirements are subject to modifications, after some tuning to the property model, exploiting simulation becomes possible to understand what are the changes to implement to make the system compliant with the new requirements.

Since the implementation of the proposed approach is library-based, it is easy to use and make reusable property models to enable the simulation-based verification. Also, the possibility to use the visual modeling tools offered by the Modelica platform supports system engineers to define the models. These are clear and remarkable advantages. Nevertheless, we can also observe some drawbacks: (i) the implementation is tool-dependent; (ii) the possibility to model requirements is limited to the available functions and blocks provided by the library. As the library implements a subset of the FORM-L constructs, theoretically not all the requirements can be modeled, even if in some cases the problem can be bypassed providing a different definition of the same requirement: e.g. if a variable is not derivable directly in the property model, it could be measured in some way from the simulation model.

According to the last motivation, a desirable feature is to improve and extend the *Modelica_Requirements* library to implement a wider set of FORM-L constructs. To understand what are the constructs that should be introduced and what the library misses, a viable solution is to extensively experiment the presented solution in different domains.

There are two critical points of the proposed approach, on which future researches could make significant advancements: the first is to support the creation of property models and the FORM-L representation, while the second is related to the binding.

About the building of property models, an improvement would be to have an automatic transition from the requirements defined in natural language to the FORM-L representation. For instance, a tool that given the four questions in natural language could translate them into FORM-L. Also, the transition from a property model, defined with blocks of the *Modelica_Requirements* library, to a FORM-L representation could be automated. Having a tool-independent representation allows the use of model checkers to perform formal model verification, with the end of understanding if there are conflicting requirements. In fact, the formal representation alone is not sufficient for this purpose, a suitable tool is required. A civil aircraft has about one million requirements, some of them are likely to be in contrast, it is valuable to identify them before to proceed to the design.

About the binding, to have a one-tool solution which allows a seamless coupling of the behavioural model to the requirement blocks can be considered the ideal situation. Usually the system and the property model are defined in heterogeneous environment: in this case the opportunities of interaction are determined by the binding solutions offered by the property model. To perform the co-simulation, the different tools could be compliant to the FMI standard [12]. To use the data exchange, the same data interchange format must be adopted. In some cases, the variables and signals needed by the property model could not be directly provided by the behavioural model, because of assumption on the measurements or how it is made the model itself. In this case the approach suggests using the concept of observer, which are function or transformation properly defined to compute the desired output from the observable measurements (e.g. given a model that generates measurements about temperature, pressure and volume one could apply a law to derive the entropy).

In future works it is expected to identify possible solutions to improve the usability, integration and results analysis capabilities of the exploited tools, e.g. with the Siemens PLM software Teamcenter. Also, the association between test scenario and results it is an important step, as well as the test scenario generation.

ACKNOWLEDGMENT

Authors would like to thank Elias Allegaert for his precious collaboration and advice on this research.

REFERENCES

[1] F. Aiello, A. Garro, Y. Lemmens, S. Dutré, “Simulation-based verification of system requirements: an integrated solution”, Proceedings of the 2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC 2017), Calabria, Italy, May 16-18, 2017.

[2] E. Allegaert, “Enabling Simulation-Driven Architecture-Based Design for Load Analysis of Complex Systems”, unpublished master’s thesis, Delft University of Technology, Delft, The Netherlands, 2017.

[3] A. Garro, A. Tundis, D. Bouskela, A. Jardin, N. Thuy, M. Otter, L. Buffoni, P. Fritzson, M. Sjölund, W. Schamai, H. Olsson, “On formal cyber physical system properties modeling: a new temporal logic language and a Modelica-based solution”, Proceedings of the IEEE International Symposium on Systems Engineering (IEEE ISSE 2016), Edinburg, Scotland, UK, October 03-05, 2016.

[4] P. Jalote, An Integrated Approach to Software Engineering, 2005, Narosa Publishing house.

[5] A. Jardin, D. Bouskela, T. Nguyen, N. Ruel, E. Thomas, R. Schoenig, S. Loembé, and L. Chastanet. “Modelling of system properties in a Modelica framework,” Proceedings of the 8th International Modelica Conference, Dresden (TU), March 2011T.

[6] Modelica Association - https://www.modelica.org/

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

[8] Nguyen, “FORM-L: A MODELICA extension for properties modelling illustrated on a practical example”, Proceedings of the 10th International Modelica Conference, Lund (Sweden), March 2014.

[9] M. Otter, N. Thuy, D. Bouskela, L. Buffoni, H. Elmqvist, P. Fritzson, A. Garro, A. Jardin, H. Olsson, M. Payelleville, W. Schamai, E. Thomas, A. Tundis “Formal requirements modeling for simulation-based verification”, Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015.

[10] Siemens PLM Automation Products (Virtual.Lab Motion, Imagine.Lab Amesim): https://www.plm.automation.siemens.com/it_it/products/lms/

[11] SysML: http://www.omgsysml.org/

[12] Functional Mock-up Interface (FMI): http://fmi-standard.org/

[13] C. Dickerson and D. N. Mavris, Architecture and principles of systems engineering (CRC Press, 2009).

[14] Wiley, INCOSE “Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities” (JohnWiley & Sons, 2015).

[15] Kapurch, Stephen J., “NASA systems engineering handbook”, Diane Publishing, 2010.