Modelling of System Properties in a Modelica Framework

Audrey Jardin Daniel Bouskela Thuy Nguyen Nancy Ruel
EDF R&D, STEP Department
6 quai Watier, 78401 CHATOU Cedex, FRANCE
audrey.jardin@edf.fr daniel.bouskela@edf.fr nancy.ruel@edf.fr

Eric Thomas Laurent Chastanet
Dassault-Aviation
78 quai Marcel Dassault Cedex 300, 92552 St CLOUD Cedex, FRANCE
eric.thomas@dassault-aviation.com laurent.chastanet@dassault-aviation.com

Raphaël Schoenig Sandrine Loembé
Dassault-Systèmes
10 rue Marcel Dassault, 78946 VELIZY-VILLACOUBLAY Cedex, FRANCE
raphael.schoenig@3ds.com sandrine.loembe@3ds.com

Abstract

In order to improve the engineering processes and especially the corresponding verification and validation phases, this article deals with the modeling of system properties in a Modelica framework. The term "property" is intended here to be generic and refers to a system requirement or limitation as well as a validity domain of a model. The choice of the Modelica language is justified by a desire to use its equation-based feature to model system properties in an unambiguous and explicit way. Besides, choosing only one formalism to describe the system properties and the physical equations of the model should ease the expression of the model validity domains.

After having introduced several theoretical concepts to formally describe a system property, the development of a dedicated library is explained and illustrated on an industrial example taken from the aeronautics domain. Some checks of system properties are thus performed by co-simulating behavioral and properties models. Finally, some extensions of the Modelica language are advocated in order to improve the applicability range and efficiency of properties modeling for complex systems, and especially to increase the rigor of their validations by enabling formal proofs.

Keywords: Modelling;Checking;Property;Modelica.

1 Introduction

The study of performance and safety is today of prime interest when designing complex systems. At each stage of the design cycle, engineers should check the conformance of their technical choices with respect to the initial specifications. In such a Verification & Validation (V&V) process, the modeling and the verification of system properties are thus a key activity. They enable to validate the chosen implementation of the system but they also ease the capitalization on the knowledge of the system. Formalizing the requirements allows to enhance the documentation of the engineering processes by keeping track of design improvements, model refinements, changes in the safety/operational expectations, and so on.

The difficulty of such a V&V approach lies, however, in the fact that, if some techniques and languages exist today to handle system properties, they often involve specific models different from the reference engineering model (i.e. the model commonly used to predict the physical behavior of the system, that is the behavioral model). Such heterogeneity may lead to some errors and thus to flaws in the proof of safety or performance.

The modeling and the checking of system properties concern industries in charge of the design of new products (e.g. automotive, aerospace) as well as the ones responsible for the operation of long-life products and faced to retrofit due to some material obsolescence and changes in operational constraints (e.g. energy producers). The properties model and the reference engineering model should thus apply for a system involving several physical domains.

For the behavioral model, the increasingly use of the non-proprietary Modelica language [1]-[2] in various industries testifies of the Modelica efficiency to conveniently describe multi-physics behaviors. Besides, thanks to its equation-based and acausal features, the Modelica language appears well-suited to build models reusable and adaptable to the different steps of the engineering cycle.

The objective of this article is thus to study to what extent the properties of a system can be modeled and checked in a Modelica framework.

A similar approach is currently ongoing, within the ITEA2 OpenProd project [3], by linking a Modelica behavioral model to a UML properties model [4]. It actually implies the development of the so-called ModelicaML UML profile [5]-[6]. However the work presented here has been performed within the ITEA2 EuroSysLib project [7] via a collaboration between EDF, Dassault Aviation, Dassault Systèmes and DLR. It takes a different point of view in the sense that the modeling and the checking of system properties are studied in a fully Modelica-based environment. This choice can be explained by a desire to reuse:

- the equation-based feature of Modelica to model properties in a more formal way;
- the same formalism as the one chosen to describe the physical equations of the models in order to ease the expression of their validity domains (which are actually a specific kind of property).

Section 2 clarifies the concept of “property” with no reference made to the way it can be implemented in Modelica. It defines what is a property and sums up the different types of properties. It also specifies the users requirements regarding properties modeling, checking and visualization.

Section 3 aims at formalizing the way a property can be modeled. Like in a formal Property Specification Language [8], the idea is to introduce some theoretical concepts especially useful to express a property in an unambiguous and explicit form. Some notions like “space/time locator”, “state” and “event” are depicted and a list of “operators” to build several types of system properties is given and illustrated on realistic examples.

Section 4 focuses on the technical implementation of these concepts in Modelica. The development of a dedicated library is explained and illustrated on an industrial example taken from the aeronautics domain. The assessment of some system properties is in particular made by simulation using Dymola [9].

Section 5 advocates the extension of the Modelica language in order to improve the applicability range and efficiency of properties modeling for the validation of complex systems.

2 Properties modeling and checking

As mentioned above, the following sub-sections are intended to set up the framework of the study. Independently of the way it can be implemented in Modelica, they are intended to clarify the concept of “modeling and checking properties” and to show its potential use in an industrial context. Notions like a “properties model” or a “behavioral model” are in particular introduced.

2.1 What is a property?

Definition 1: A “property” is an expression that specifies a condition that must hold true at given times and places. It results in a Boolean variable stating whether the property is satisfied or not.

A property may thus specify:

- an **allowed operating domain** the system must not leave for safety reasons;
- an **operational domain** where, for instance, the system operation is optimized for performance;
- the **validity domain of a model** outside of which the corresponding behavioral equations are no longer valid;
- …

Example: Some realistic properties can be formulated in a textual form such as:

- The power plant should evolve in an allowed (temperature, pressure)-domain;
- Cavitation should never happen in a pump component;
- The characteristics of the pump are only valid for a given range $\left[Q_1;Q_2\right]$ of flow rates.
- …

Different categories of properties may actually be distinguished. A first typology may be drawn depending whether the properties are associated with the system, a sub-system or a component. But, the properties may also be classified depending on the kind of expectations. Two main kinds of properties may however be highlighted:

- a first kind where the properties characterize the expectations of the designer but also the limitations of the chosen system, sub-systems and components. These properties are expressed independently from any behavioral model;
- a second kind where the properties define some validity domains and are thus attached to specific behavioral models. These properties do not belong to the designer requirements. They only reflect how the designer represents the implementation of its system.

From the tool and language perspective, modeling system and components properties and expressing validity domains are however essentially similar. For the sake of simplicity, the term of "property" will then be used all along the paper to refer to any requirement/limitation the engineer wants to express on its system/sub-system/component or on its model/sub-model.

2.2 Uses of properties modeling

The modeling and the checking of properties may be used in an industrial context to verify and validate each stage of the system development cycle, in particular:

- to enhance the documentation of the system regarding the description of the expected behavior as well as the description of the assumptions made during the modeling of its behavior. This may in particular be useful to ease the capitalization and the transmission of knowledge;
- to improve the engineering processes by expressing the requirements in an explicit and unambiguous form and by keeping track of any evolutions due to design improvements, to changes in operational expectations, to model refinements…

Once the properties have been modeled, a series of tests can then be performed:

- to check the coherence and the completeness of the requirements (e.g. by formal proofs and consistency checks);
- to verify the conformance of the designed system with respect to the initial specifications (e.g. by simulating both the properties model and the behavioral model);
- to validate the Instrumentation & Control (I&C) part of a process on the basis of the services it should provide to the physical process, during the specification phase, and after the programming phase using hardware-in-the-loop;
- to support advanced modeling approaches like scenarios simulating sequential changes of different operating modes (e.g. simulation of a system entering a dysfunctional mode).

2.3 Distinction between a "behavioral equation" and a "property expression"

Behavioral equations describe a potential implementation of the system at the design phase, or how the system actually works during operation. They are based on physical or empirical laws.

Properties define what the system should guarantee, or in other words what is the validity domain of the system's behavior. They can also be used to define the validity domain of the model used to represent the system's behavior.

Example: Let us consider a valve. A behavioral equation can be "$\Delta P(t) = k / \rho \cdot Q^2(t)$" (where $\Delta P(t)$ is the pressure loss across the valve, ρ is the fluid density, k is the pressure loss coefficient of the valve and $Q(t)$ is the mass flow rate through the valve) whereas a property can be "For all t, $\Delta P(t)$ should be greater than $\Delta P_{\min}$ to avoid cavitation".

This distinction is important because behavioral equations and properties are fundamentally different:

- They correspond to **different objectives**: behavioral equations describe **how** the system actually works (e.g. the dynamics of the system) whereas properties define **what** the system should do (e.g. which services it should provide, the prescribed operation domain….);
- They are of **different natures**: behavioral equations define system characteristics which are always localized to a specific part of the system, whereas properties define system characteristics which may be global in time or space, in the sense that they can constrain variables across several periods of time and different locations;
- They involve **different expertise**: writing behavioral equations requires expertise in physical system modeling, whereas defining properties requires expertise in system operation;
- They have **different lifecycles**: the definition of properties occurs during the requirement phase, whereas the modeling of the system's

behavior occurs during the design phase. For instance, properties may capture the current safety rules, while behavioral equations may describe the current behavior of the system under operation. The impact of the evolution of safety rules on the operation of the system may be assessed by modifying the properties and checking them against the current system's behavior. Inversely, the compliance of system's modifications wrt. the current safety rules can be checked by comparing the modified behavioral equations wrt. the current system's properties.

2.4 Requirements on properties modeling

A property has to be:

- **in interaction with** the behavioral model of the system (since its satisfaction depends on the evolution of the system);
- **transparent** wrt. the dynamic evolution of the system (should not influence the evolution of the system);
- **coherent** with the behavioral model by not implying a too low level of details (properties should not refer to characteristics that are not depicted in the behavioral model);
- **readable** for the sake of documentation and transmission of knowledge;
- **understandable** to ease the interpretation of its potential failure.

The modelling of properties must then be in accordance with these different axioms and an adequate **data model** has to be established in particular to guarantee the transparency of the properties model towards the behavioral model.

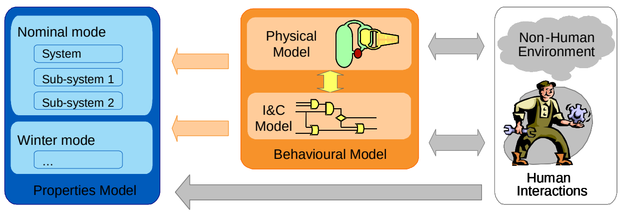


Figure 1: Data model for modeling the system behavior, the evolution of its environment and the properties the system must guarantee

So as to bound the properties model with the behavioral model in such a way that they remain dissociated, we suggest here to physically separate these two kinds of models in two kinds of files. Such a data model (Figure 1) thus corresponds to a model organized in three different parts:

- the **environmental model** where the characteristics and the evolution of the system environment are specified. This part may in particular be used to set the inputs of the simulation and so to specify some scenarios (e.g. introduction of some component failures, simulation of a series of operator intervention,…)
- the **behavioral model** where the intrinsic characteristics and the evolution of the system are described with behavioral equations. In other words, this part corresponds to the physical modeling of the process and its I&C part;
- the **properties model** where the expected services of the system and the validity domain of the behavioral model are depicted.

In order to ensure the fact that the properties model should be only an observer of the behavioral model, the three parts of this data model must communicate with each other such that:

- the properties model and the behavioral model may access the data described in the environmental model (the properties as well as the behavior of the system may actually change depending on the evolution of the system environment);
- the properties model may access the data described in the behavioral model in order to evaluate whether the dynamic evolution of the physical process and its I&C part stay within the bounds of the prescribed properties domain, but it cannot send any data to influence the behavioral model.

Besides, in order to ease the reading and the construction of the properties model, it may be helpful to organize it into a hierarchy. Depending on the modeler expectations, this hierarchy may be based:

- on the architecture of the studied system;
- on the different states of the studied system and its environment;
- or on a combination of the system architecture and the different states (as in Figure 1).

A hierarchy based on the system architecture may be useful in particular when the architecture of the system changes and the modeler has then to remove or to add some properties related to some specific subsystems or components. On the other hand, a hierarchy based on the states of the system and of its environment may add further information on how the system should behave (the description of these states gives in general a better insight into the different operating modes).

In practice, since a different properties model can be built for each operating mode, several properties models can be associated with the same behavioral model.

An advanced data model has thus been imagined to enable the handling of such a situation. As shown in Figure 2, our suggestion is to add a statechart model [10] where the different states of the system and of its sub-systems and components are described. The main idea is then that the statechart model is viewed as a supervisor: it may access the data of the behavioral model, decide in which states the system operates and select the appropriate properties model.

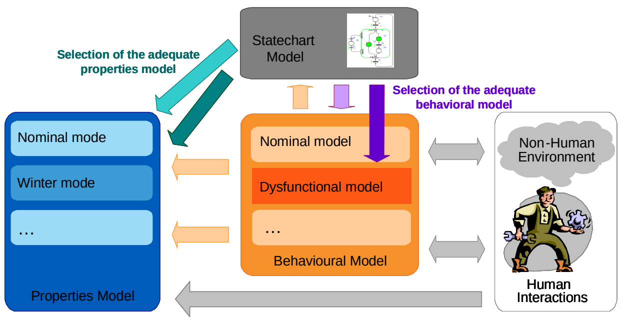


Figure 2: Advanced data model for switching properties models and behavioral models

With the same point of view, it may also be useful to associate several behavioral models with the same system. For instance, if the objective is to anticipate the physical behavior of the system when a fault occurs and to verify whether the corresponding behavior remains in a safe domain, it may be helpful to switch, during the simulation, between a model describing a nominal behavior and another model describing a dysfunctional behavior of the system.

For this particular use, the advanced data model of Figure 2 can be adapted in such a way that the statechart model may:

- access the results obtained from the assessment of the properties to detect if the validity domain of the active behavioral model has been crossed;
- activate, if needed, another behavioral model with an appropriate validity domain.

In such application, let us note however that even if the same statechart model supervises several properties and behavioral models, the hierarchy of the properties may not necessarily correspond to the hierarchy of the behavioral models.

2.5 Requirements concerning the checking, the visualization and the analysis of a property

The question now is to study to what extent the properties and the behavioral models should be coupled together to check whether the properties are satisfied or not.

Two kinds of checks may be imagined: a static check by formal proof and a dynamic check by simulation.

Checks by formal proof can be used to verify the coherence: (1) between the properties themselves (e.g. to verify that properties are compatible between themselves and do not define mistakenly empty operating domains); (2) between the properties and the behavioral model (e.g. to check that the behavioral equations are mathematically compatible with the properties).

Complementarily, **checks by simulation** can be used to verify the properties all along a given scenario such as the “Virtual Verification” method suggested in [11].

Checks by formal proof require that properties and behavioral models are described using **high level formal declarative languages** such as Modelica. Checks by simulation require the **definition of scenarios** with possible occurrences of dysfunctional modes, injection of faults, changes because of human interactions, and so on. Besides, to help the analyst understand the reasons of properties violations, **diagnostics tools** should be provided, such as:

- generation of alarm when a property is violated: a pop-up may appear during the simulation as soon as the non-verification of a property is detected;
- change of component’s visual aspect: the color of a component may change when its corresponding properties are not satisfied;
- edition of a log file: to recap all the properties violations and to signal at first glance when and where the problems have appeared;
- properties filtering: the analyst may need to make a distinction between safety properties and properties indicating some pre-alarms, optimal operating domains, constraints avoiding damages, and so on. The possibility to tag the properties with adequate flags may, for instance, be considered.

This list is however far to be exhaustive, for instance one can also imagine the possibility to introduce some indicators like the probability of a property’s failure.

3 Theoretical concepts used for properties modeling

By definition and similarly to the Behavior Engineering approach [12], a property can be expressed under the generic form: [Where][When][What].

Example: The avoidance of pump cavitation can be expressed by: [In Pump1][for every instant when the pump operates][the fluid pressure at the inlet should be greater than a minimum value].

To meet one of the requirements which is to formalize the expression of the properties, the main idea of the following sub-sections is namely to describe the concepts related to the generic form of a property. In more details, these sub-sections describe what are the "attributes of a property" (i.e. what refer to the where, when and what terms), what they imply (i.e. what "kinds of objects" are used) and how they can be built (i.e. which "operators" are used to construct such attributes).
In order to set up a clear theoretical framework, the following paragraphs are still voluntarily independent of any dedicated language. The implementation in a Modelica environment is studied in Section 4.

3.1 Attributes of a property

As already stated, a property may be expressed under the generic form [Where][When][What].

"Where" is a **space locator** that specifies which part of the system is concerned by the property. The space locator specifies a subset of "everywhere". It may involve a family of components (e.g. all the pumps), a specific component (e.g. pump n°2), a part of a component (e.g. a specific segment of a pipe), a subset of objects that are in a given state or that are satisfying a given condition (e.g. all the components whose temperature exceeds 240°C). It may also be a subset or a combination of other space locators.

"When" is a **time locator** to indicate at which instants the property has to be satisfied. The time locator is a subset of "always". It may involve a time instant referring to all the occurrences of a specific event (e.g. when a pump starts), a time period during which a given condition holds true (e.g. as long as the pump operates), a sliding time interval when a given property needs to be satisfied only most of the time (e.g. for no more than 3 minutes over any period of 2 hours). It can also be a combination of several time locators.

"What" refers to the **condition** the system should guarantee (or the assumption the model should satisfy in the case of a validity domain). It consists in an expression that can be evaluated and which results in a Boolean variable stating whether the property is satisfied or not. Because of the variety of properties, conditions can involve physical variables and/or states probed at specific time instants or during specific time periods. They may also imply events, or even a combination of these several kinds of objects with some space and time locators.

3.2 Types of objects implied in the attributes of a property

As shown above, a property may entirely be defined by the association of a space locator, a time locator and a condition to be satisfied. Due to the complexity of the systems and the different types of properties the designer is interested in, these attributes have to deal with numerous kinds of objects such as:

- **instances of models** (e.g. Pump1, SensorMT018…);
- **geometric data** (e.g. segment[0.2…0.8] of Pipe3);
- **physical variables** and/or parameters of different physical types;
- **states** of the system, sub-systems and components (e.g. since the expected services are often depending on the different operating modes, a property may concern a component only when it is not in a dysfunctional state);
- **events** that characterize external stimuli of the system (e.g. human intervention, evolution of the system's environment) or internal changes of the sub-systems and their components (e.g. fault during a valve opening);
- combinations of instances of models, geometric details, physical variables, states and events built thanks to some specific **operators** (e.g. Pump1 and Pump2, for every instant where Pump1 operates, all pumps except Pump3…).

The two following paragraphs define in more details what we mean by the notions of "state" and "event" which may be less naturally intuitive. The concept of "operator" to build adequate properties attributes is then studied in Section 3.3.

3.2.1 State

Definition 2: A "state" is a discrete variable that characterizes an aspect of a system, a sub-system or a component. It can take its values only within a finite set of enumerated values. It is defined according to a specific point of view on the system.

Example: States may correspond to different operating modes (e.g. maintenance/normal operation), operating conditions (e.g. cold winter/hot summer), physical behaviors (e.g. pump/turbine mode)…

A state may have one (or several) attached sub-state variable(s) and the same system, sub-system or component may have multiple independent states.

As explained above, the notion of state can also help to organize the properties and the behavioral models into a hierarchy.

3.2.2 Event

Definition 3: An "event" is an object that is generated at a given time instant to signal the occurrence of a fact. It carries at least two pieces of information: the date and the class of the event. The former indicates when the change has occurred while the latter defines what has happened. An event has no duration and does not characterize what are the consequences of the change on the system behavior.

Example: The starting of a motor may generate an event.

In some cases, sub-classes of the event concept may be created to provide further information such as a probability distribution, a frequency of appearance, and so on. A distinction may also be made depending on the location of the change. For instance, internal events are related to the evolution of some variables in the behavioral models while external events correspond to changes in the system's environment.

The introduction of the event concept can especially be used to define some simulation scenarios with injections of faults, control or perturbation actions.

3.3 Operators to build property attributes

An operator is defined here as a function that constructs an object as output given one or several objects as inputs. Inputs and outputs may be of the same type, or of different types. Operators are of prime importance to build space/time locators or even conditions:

- when a simple observation of the variables available in the behavioral model is not sufficient to describe what the system should guarantee or when the model is valid (e.g. when a behavioral model involves only a mass flow rate variable and the corresponding property is expressed in terms of volume flow rate, an operator has to be used to perform the unit conversion);
- or when it is easier to express it as a function or a combination of other attributes.

Operators may be classified depending on the types of their inputs and outputs. From the analysis of industrial needs based on EDF and Dassault-Aviation use-cases, some operators have been identified as particularly useful, such as:

- **arithmetical operators** and usual functions: +, -, *, ÷, cosinus, absolute value,…;
- **logical operators**: and, or, …;
- **set operators**: all, in, $\in$, $\notin$ (creation of a set), $\cup$ (sets union), $\cap$ (sets intersection), $card(\ )$ (cardinal of a set), $\overline{S}$ (complement of a set), \ (subtraction of a subset), …;
- **operators on time and events**: for, when, while, always, never, delay between two events, duration of a time period, count of the number of events, events synchronization;
- **dedicated operators**: >, <, ≥, ≤ (thresholds), $\subset$, $\not\subset$ (domain inclusion), Δ (ramp), A (accumulation), $freq(\ )$ (frequency).

Among this (non-exhaustive) list, some operators refer to well-known concepts in mathematics or computer programming, while other correspond to operators more specific to the modeling of properties for physical systems. The aim of the following sections is to give a better insight into these dedicated operators by furnishing their mathematical description and illustrating their uses. Their implementation in Modelica will then be further discussed in Section 4.

3.3.1 Threshold operator

Definition 4: The "threshold operator" defines a lower (or an upper) limit that a variable should not exceed.

This operator may be used to build a time locator or a condition.

Examples: In the case of an heat exchanger, an external leakage may appear if the pressure and the temperature both exceed given maximum values. An internal leakage may also occur if the number of cycles is superior to a specific limit. Air bearing can be destructed if its rotational speed crosses a maximum value. If the Mach number is superior to a specific limit, the model of an air pipe may predict pressure losses with less confidence.

3.3.2 Domain inclusion operator

Definition 5: The "domain inclusion operator" defines a continuous and delimited area that variables should not go beyond. It can be seen as the generalization of the threshold operator to the multi-variable case.

This operator may be used to build a condition. The delimited area may be defined by a set of given values or an analytical expression.

Examples: Figure 3 shows different (pressure ratio, reduced airflow) domains for a centrifugal compressor: the green area stands for the nominal operating domain whereas the orange and the red ones denote respectively the restricted and the destructive domains.

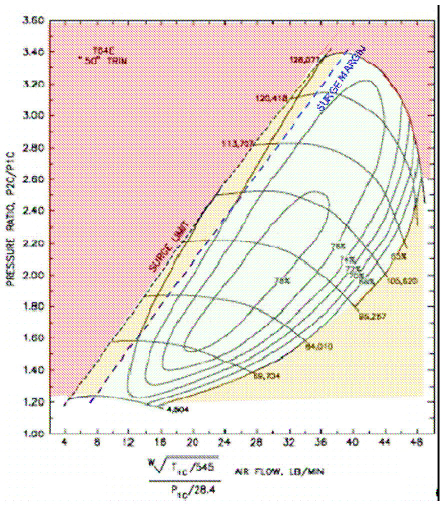


Figure 3: Example of a (pressure ratio, reduced airflow)-map for a centrifugal compressor

3.3.3 Operator for monitoring a rate of change

Definition 6: The "ramp operator" Δ can be defined both on intensive and extensive variables. For an intensive variable $G_i(t)$ (resp. extensible variable $G_e(t)$), it provides the evolution of $G_i(t)$ (resp. $G_e(t)$) per time unit during the time period $[t_0,t_1]$ such as:

$$\Delta(G_i,t_0,t_1)=\frac{G_i(t_1)-G_i(t_0)}{t_1-t_0}$$

$$\Delta(G_e,t_0,t_1)=\frac{G_e(t_1)-G_e(t_0)}{G_e(t_0)\cdot(t_1-t_0)}$$

Examples: In a heating system, the temperature increase should not exceed 3°C per hour. The power should not increase more than 2% per hour. For an aircraft, the cabin altitude rate of change should correspond to certain pressure bump duration (Figure 4).

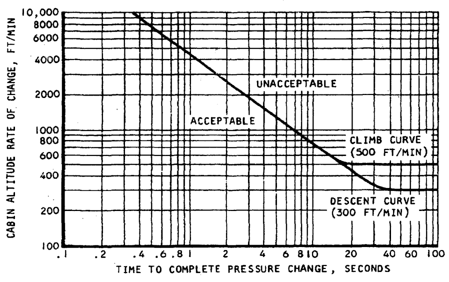


Figure 4: Example of a (cabin altitude rate of change, pressure bump duration)-map to ensure passenger comfort in an aircraft

3.3.4 Operator for monitoring a time integration

Definition 7: The "accumulation operator" is relevant for extensive variable only. For an extensive variable $G_e(t)$, its accumulation during the time period $[t_0,t_1]$ is defined such that:

$$A(G_e,t_0,t_1)=G_e(t_1)-G_e(t_0)=\sum_k\int_{t_0}^{t_1}\Phi_k(G_e)\cdot dt$$

where $\Phi_k(G_e)=\int_A \rho\cdot g_e\cdot v\cdot dA$; ρ is the mass density; g_e denotes the variable $G_e(t)$ per mass unit and v is the velocity through the surface A of the control volume.

Examples: This operator can be used to monitor the cumulative radioactive dose which is emitted by a sub-system in a power plant. It may also prevent a complete clogging of a cold heat exchanger by setting up the condition

"$\int_{t_0}^{t_1} AirHumidity\cdot MassAirFlow\cdot dt$ should be inferior to a given value of mass".

3.3.5 Operator for monitoring oscillations

Definition 8: The "frequency operator" allows to quantify the time period between the occurrence of some identical events.

Examples: For a regulating valve, the frequency of its control signal should not exceed, for instance, 1 Hz during more than 30 seconds in order to avoid any impact on the valve's Mean Time Between Failure.

3.4 Construction of a property

To recap the theoretical concepts introduced above, we can state that a property can be formally described as the association of three attributes (namely a space locator, a time locator and a condition to be satisfied) which are built through specific operators from objects provided by the behavioral model(s).

The following examples serve as an illustration on how a property expressed in a textual form may be reformulated in a formal manner. Through some combination processes, they also show how complex properties may be entirely described from the short list of formal concepts previously described.

Examples: To avoid the Pump1 cavitation, the pressure at the inlet should be greater than a minimum value → [In Pump1] [for all t | state = operating] [condition P(t) > P_{min}].

To ensure the performance of the cooling system, at least two pumps should be operating → [In Cooling-System] [Always] [condition cardinal (set (Pump | state = operating)) ≥ 2].

To avoid the turbine wheel erosion, no liquid water should enter at the inlet during more than 30 minutes → [In Turbine][Always][condition duration (Inlet-Water.state = liquid) ≤ 30 minutes].

4 Modeling of properties in a Modelica framework

First, a Modelica library dedicated to the modeling of properties that has been developed during the EuroSysLib project is presented and illustrated on an industrial use-case. Then, the current limitations of this library are discussed. Finally, a rationale for a Modelica language extension to support properties checking by formal proof is given.

4.1 Modeling of properties with a dedicated Modelica library

4.1.1 Purpose of the library

The aim of the library is to make checks on parts of an architecture defined by a Modelica behavioral model. Its particular features are the following:

- It enables checks during simulations, gets and stores information in case of detected defect for actions (e.g. stops the simulation and starts the next one according to specified criteria);
- It enables reuse of the properties by parameterizing them according to the potential uses of the model (e.g. mission profile, specific boundary or environmental conditions…), stores the properties in a catalog for reuse;
- It enables dysfunctional analysis: check of properties must not influence model simulation (e.g. potential change of time step coming from properties evaluation must not lead to an unwanted decrease of results accuracy). But, properties could be used to change the behavior of models with defect (detected by properties observers). In this case models should be modeled with different behaviors which could be activated on demand (with currently smooth change between behaviors due to the change of the equation structure).

The major particularity is that checks are not done as post-processing, but on the fly at run-time.

4.1.2 Use-Case: Environmental Control System (ECS)

The simple ECS used for testing properties is defined by two main parts: (1) the Cold Air Unit (CAU), made of pipes, heat exchangers, compressors and turbines, and which controls air characteristics provided to the cabin and bays; (2) the bleed, which provides air from the engines to the CAU (Figure 5).

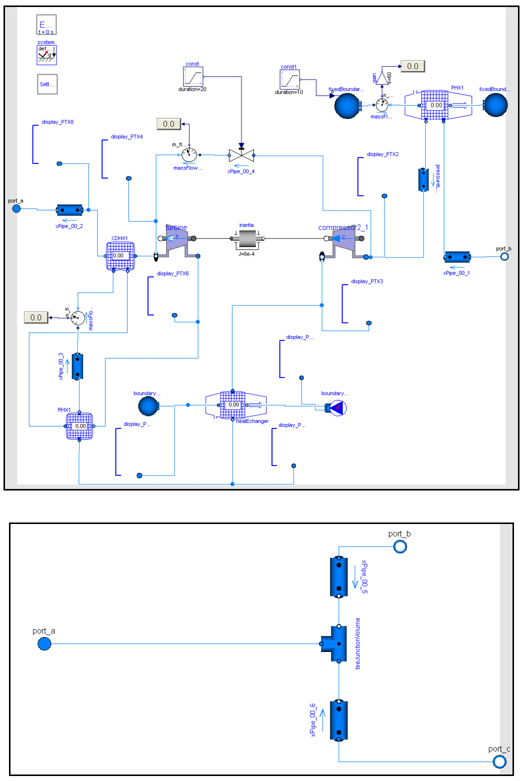


Figure 5: Cold Air Unit and Bleed of the Environmental Control System

The ECS must be compliant with many requirements. These requirements are classified according to 6 categories:

1. **threshold monitoring**, which deals with crossing of threshold (see paragraph 3.3.1 for examples of properties on the heat exchanger, air bearing and pipes);

2. **operating domain monitoring**, which mainly deals with conditions regarding the location of couple of values (or more) inside a defined area or inside a volume. Currently only requirements regarding 2D area is defined here, but conditions with more than two dimensions could occur (see paragraph 3.3.2 for example of allowed domains for a compressor);
3. **rate of change monitoring**, which deals with conditions with derivatives (see paragraph 3.3.3 for criteria concerning the cabin altitude rate of change);
4. **accumulation monitoring**, which deals with conditions with integration (see paragraph 3.3.4 for example of a condition to prevent the clogging of the exchanger);
5. **oscillation monitoring**, which deals with conditions based on oscillation characterizations (occurrences, frequencies) (see paragraph 3.3.5 for the example of a regulating valve);
6. **monitoring with space/time locators**, which test conditions linked to location of components or events (see paragraph 3.4 for the example of a property to avoid turbine wheel erosion).

4.1.3 Structure of the library

The presented library is currently dedicated to the ECS use-case (with behavioral components developed by Dassault-Aviation) and properties observers.

This use-case appears here as a library divided into two main parts (Figure 6):

1. A generic part, called “PropertyObservation”. It contains:
   a. examples, especially models from DLR, which propose two ways (a direct link or a bus) for connecting the properties to the ECS model.
   b. the ECS model using the second type of properties connection since it appears as the most generalized and readable way for complex systems involving numerous properties.
2. A second part which contains use-cases. It is split into models and requirements:
   a. The models are here focused on an ECS system simulated with dry or moist air;
   b. The properties are a collection of specific properties built as generically as possible and classified into several types of properties.

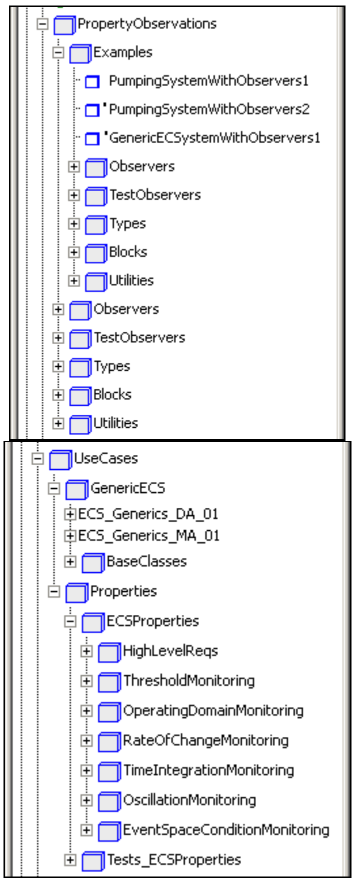


Figure 6: The two main parts of the library

4.1.4 Process for properties modeling and checking

4.1.4.1 Connecting models to properties

When a system must be checked regarding its compliance with requirements, a good process is to integrate the model inside a virtual test bench by extending it. In this way the model can be checked according to several sets of requirements.

An expandable bus called RequirementBus is added to the models by drag and drop from the library. Requirements or sets of requirements are then connected to the RequirementBus. Parameters of the requirements can be adjusted according to the analysis. All values which must be provided by the model are then automatically available on the bus.

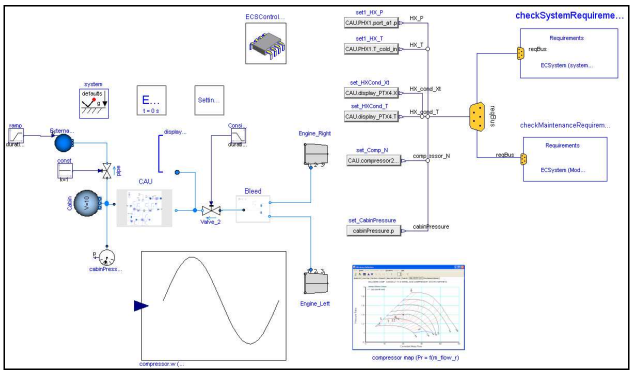
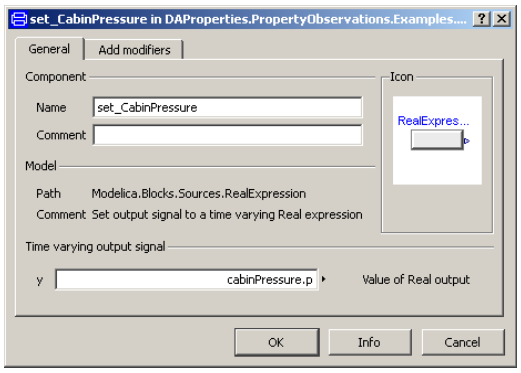


Figure 7: ECS use-case

All the necessary interfaces must then be defined one by one using components called Modelica.Blocks.Sources.RealExpression from the Modelica Standard Library.

The component RealExpression must be linked to a variable in the model as in Figure 8 for the cabin pressure.



set_CabinPressure in DAProperties.PropertyObservations.Examples....

General | Add modifiers

Component

Name: set_CabinPressure

Comment:

Icon: RealExpres...

Model

Path: Modelica.Blocks.Sources.RealExpression

Comment: Set output signal to a time varying Real expression

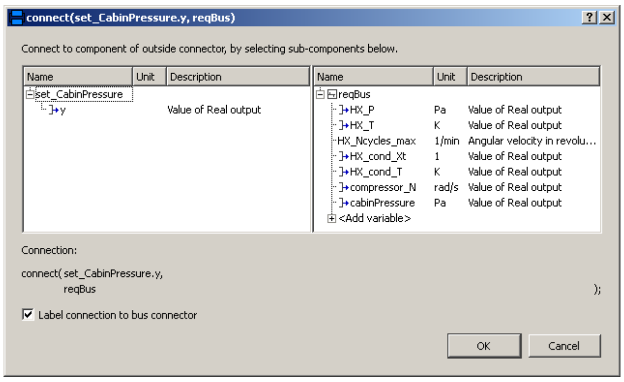
Time varying output signal

y: cabinPressure.p — Value of Real output

OK | Info | Cancel

Figure 8: Example of value of a component RealExpression

When connecting the component, a window appears for mapping variable selections (Figure 9). It allows the user to select which variable must be mapped to the component RealExpression.



connect(set_CabinPressure.y, reqBus)

Connect to component of outside connector, by selecting sub-components below.

Name	Unit	Description
set_CabinPressure		
y		Value of Real output

Name	Unit	Description
reqBus		
HX_P	Pa	Value of Real output
HX_T	K	Value of Real output
HX_Ncycles_max	1/min	Angular velocity in revolu...
HX_cond_Xt	1	Value of Real output
HX_cond_T	K	Value of Real output
compressor_N	rad/s	Value of Real output
cabinPressure	Pa	Value of Real output
<Add variable>		

Connection:

```
connect( set_CabinPressure.y,
         reqBus                                                    );
```

☑ Label connection to bus connector

OK | Cancel

Figure 9: Window appearing for connecting RealExpression to input variable within Requirements

4.1.4.2 Hierarchical decomposition of properties

Requirements may be complex with many elements. Therefore putting all requirements at the same level may be cumbersome.

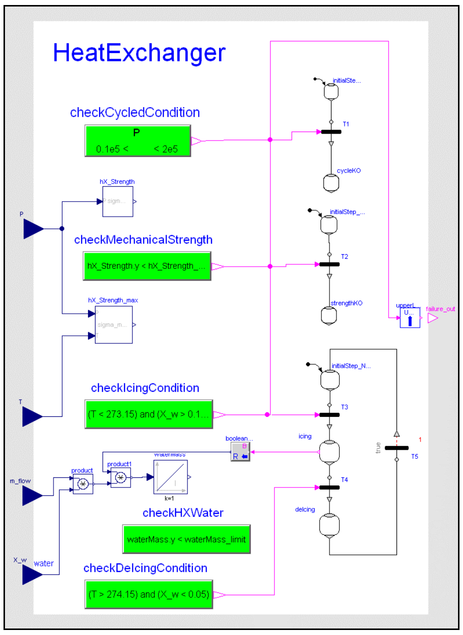


Figure 10: Requirements for a heat exchanger

A simple example on the ECS heat exchanger is shown in Figure 10 where the properties are composed of three main conditions:

- **checkCycledCondition**: counts the number of pressure cycles seen by the heat exchanger and sets a warning when this number is upper a threshold;
- **checkMechanicalStrength**: computes an equivalent stress within the heat exchanger and compares it to an allowed maximum stress;
- **checkIcingCondition**, **checkDeIcingCondition** and **checkHXWater**: check icing and deicing conditions, and the amount of water inside the heat exchanger. Typically, if the mass of water is above a limit, the heat exchanger could be partially clogged and the simulation could be not valid if the behavioral model is not adapted to this particular situation. When attaining this operating condition, it is interesting to continue the simulation with the properly modified behavioral model to analyze the consequences of being outside of the nominal domain (dysfunctional analysis).

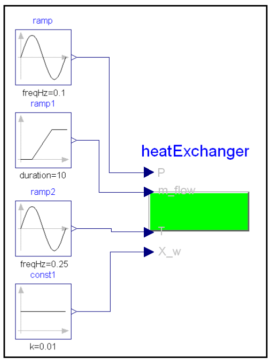
These requirements stand for the heat exchanger but all the types of properties defined in Section 3 have been investigated within the complete ECS use-case.

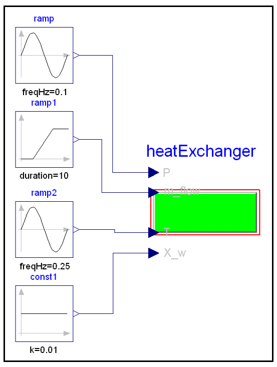
In fact, many other properties observers could be added and if we consider observers of other components or sub-systems it seems to be cumbersome to put them all in the same view. Therefore the library has been enhanced to support hierarchical decomposition of properties. In particular a component called UpperLevel has been introduced to transmit the result as an OR function of its inputs.

4.1.5 Unit test for a heat exchanger

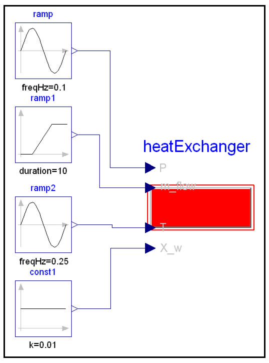
Properties components have been tested with specified inputs to check that their behaviors were correct. Figure 11 shows different properties states for a heat exchanger.

During simulation, the visual indicator stays green as long as no defect is detected (state 1). When a defect occurs, the edge of the indicator turns to red (state 2) and goes back to green as soon as the defect detection disappears. To keep the memory of a defect detected during the simulation, another outside red square is added and remains until the end of the simulation (state 3).





(a) State 1 **(b) State 2**



(c) State 3

Figure 11: Warning indicators of a set of properties

For a detailed analysis, it is possible to access the internal warning indicators of each property as shown in Figure 12.

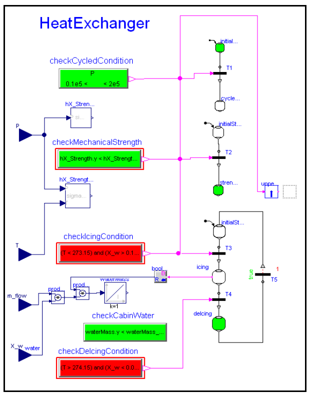


Figure 12: Internal warning obtained at the lower level of requirements (detailed level)

It is also possible to investigate more deeply what has happened by plotting all variables of interest.

5 Formal modeling of properties with Modelica language extensions

The previous section has shown that the development of a dedicated Modelica library is efficient to model the main properties implied in the ECS industrial use-case. Two current limitations have however to be mentioned.

Firstly, even simple properties cannot be modeled as soon as they imply space or time locators.

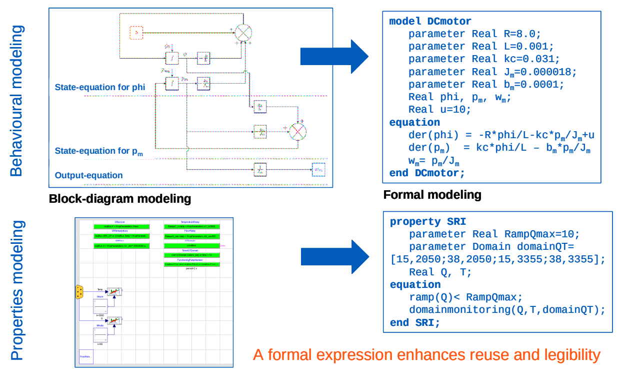
Examples : Currently the following properties cannot be modeled.

→ [In Turbine] [Always] [condition duration (InletWater.state = liquid) ≤ 30 minutes]

→ [In CoolingSystem] [Always] [condition cardinal (set (Pump | state = operating)) ≥ 2].

Secondly, even if the properties library features the all main operators needed to model properties, it only supports the construction of the properties in a block-diagram way: many components must be connected to form one simple property. This approach is quite in contradiction with the Modelica spirit as it emphasizes a graphical modeling approach over a formal equation modeling approach. Therefore, it does not comply with the fundamental requirements for the formal description of properties. This leads to several potential limitations, such as the impossibility to check properties by static proofs, or the impractical association of validity domains to behavioural equations.

As for the modeling of the physical behaviors, a formal (i.e. an equation-based) approach presents numerous advantages to model properties (Figure 13).



```
model DCmotor
   parameter Real R=8.0;
   parameter Real L=0.001;
   parameter Real kc=0.031;
   parameter Real Jm=0.000018;
   parameter Real bm=0.0001;
   Real phi, pm, wm;
   Real u=10;
equation
   der(phi) = -R*phi/L-kc*pm/Jm+u
   der(pm)  = kc*phi/L – bm*pm/Jm
   wm= pm/Jm
end DCmotor;
```

```
property SRI
   parameter Real RampQmax=10;
   parameter Domain domainQT=
[15,2050;38,2050;15,3355;38,3355];
   Real Q, T;
equation
   ramp(Q)< RampQmax;
   domainmonitoring(Q,T,domainQT);
end SRI;
```

Figure 13: Advantage of a formal (equation-based) approach for behavioral and properties models

A formal description allows to:

- provide explicit and unambiguous specification of properties and thus avoid some potential misunderstandings and mistakes;
- enhance the legibility and so the reuse of the properties models;
- improve the test coverage by automating the checking procedures;
- enable some static tests (i.e. tests performed without any simulation) on the coherence and the completeness of the properties.
- associate validity domains to behavioural equations, and perform various checks on them.

Examples: For a model where the two properties "[In Pump1][Always][condition P(t) > P_{min}]" (to avoid pump cavitation) and "[Always][condition P(t) < P_{max}]" (to guarantee the flow direction of radioactive leaks) should be satisfied, a first check should ensure that there is no contradiction between the numerical values of P_{min} and P_{max}.

In another model, if the two following properties "[For 0 °C < T < 15 °C][condition …]" and "[For 22 °C < T < 38 °C][condition …]" should be fulfilled, one may wonder if there is some incompleteness in the requirements and what should happen when the temperature is between 15 °C and 22 °C.

Hence modeling the properties with an equation-based approach will give the possibility to perform formal transformations and verifications on both the properties and behavioral models. This will contribute greatly to improve system validation by increasing the coverage and the rigor of the verifications.

The use of Modelica for that purpose may only be done by introducing natively in the language the concepts of space/time locators and dedicated operators.

6 Conclusions

In order to improve the V&V process, this article deals with the modeling and checking of system properties. The study is made within a fully Modelica-based framework and encompasses with the term "property" the modelling of any requirement/limitation the engineer wants to express on its system/subsystem/component or on its model/submodel.
Imagined as complementary to the ModelicaML approach, modelling system properties directly in Modelica is justified here as a desire to: (1) use an equation-based language to express the properties in an unambiguous way; (2) choose a formalism closed to the one used for expressing the physical equations in order to ease the formulation of the validity domains of the models.
After having introduced some theoretical concepts to formally describe a property, some requirements have been listed on how the properties and the behavioural models should communicate to check virtually whether the properties are satisfied or not.
The development of a Modelica library dedicated to the modelling of properties has then been explained and illustrated on an industrial example taken from the aeronautics domain. Even if several operators have been especially built to cover the most types of properties, two current limitations have however to be raised: (1) even simple properties cannot be modeled as soon as they imply some space or time locators; (2) the properties are actually modeled in a block-diagram way which is inconsistent with the ambition of performing formal proofs.
Further work has then to be investigated to make up for these aspects and concrete proposals should be made to introduce natively in the Modelica language the concepts of space/time locators and dedicated operators.

Acknowledgements

This work was partially supported by the pan-European ITEA2 program and the French government through the EuroSysLib project.

References

[1] Information available on the Modelica Association web site: http://www.modelica.org

[2] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, Wiley IEE Press, 944 pages, February 2004.

[3] Information available on the OpenProd web site: http://www.ida.liu.se/~pelab/OpenProd/

[4] Information available on the official UML web site: http://www.uml.org

[5] W. Schamai, P. Fritzson, C. Paredis, A. Pop, *Towards Unified System Modeling and Simulation with ModelicaML: Modeling of Executable Behavior Using Graphical Notations,* in Proceedings of the 7th Modelica Conference, Como, Italy, September 20-22, 2009.

[6] W. Schamai, *Modelica Modeling Language (ModelicaML): A UML Profile for Modelica*, technical report in Computer and Information Science, n° 2009:5, Linköping University Electronic Press, 49 pages, 2009.

[7] Information available on the EuroSysLib web site: http://www.eurosyslib.com

[8] *Property Specification Language – Reference Manual*, Accellera technical report, USA, June 2004.

[9] Dymola software, Dassault Systèmes, information available at: http://www.dymola.com

[10] D. Harel, *Statecharts: A visual formalism for complex systems*, in Science of Computer Programming, 8(3):231-274, June 1987.

[11] W. Schamai, P. Helle, P. Fritzson, C. Paredis, *Virtual Verification of Systems Design against System Requirements – A Method Proposal*, in Proceedings of the 3rd International Workshop on Model Based Architecturing and Construction of Embedded Systems (ACES 2010), in conjunction with MODELS 2010, Oslo, Norway, October 4, 2010.

[12] T. Myers, P. Fritzson, R.G. Dromey, *Seamlessly Integrating Software & Hardware Modelling for Large-Scale Systems*, in Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT 2008), Paphos, Cyprus, July 8, 2008.

[13] *Eurosyslib sWP7.1 DGT116083B Dysfunctional Use Cases and User Requirements*, 2010.

[14] *EuroSysLib sWP7.1 DGT124618 Properties Evaluation Report*, 2010.

[15] *EuroSysLib sWP7.1 Properties Modeling*, 2010.