OPENPROD

ITEA 2

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

Report R6.34

Authors: G. BRUEL, A. JARDIN (EDF)

Date: 12/03/12

I&C functional validation based on the modelling of requirements and properties: evaluation of ModelicaML

AVERTISSEMENT / CAUTION

Synthesis

The present paper corresponds to a deliverable of Work Package 6 of the ITEA2 OPENPROD project. In particular, it deals with the evaluation of the ModelicaML development made in Work Package 2 and the adequacy of this approach mixing both Modelica and SysML to validate the control systems of EDF power plants.

- The first part introduces the document.
- The second part gives an overview of EDF I&C design activities with a special focus on functional validation.
- The third part presents EDF needs regarding functional validation.
- The fourth part presents EDF background regarding model-based system validation.
- The fifth part gives a presentation of the ModelicaML environment.
- The last part of the document gives the conclusions of the evaluation work that has been performed on ModelicaML and the conceptualization work performed on structuring the test environment.

In the end, the conclusion reminds the main characteristics of ModelicaML along with the progress made in elaborating an adequate method for model-based testing and in structuring the corresponding test environment.

ModelicaML is a research project hosted by the Open Source Modelica Consortium and mainly led by Wladimir Schamai, EADS. ModelicaML is an environment comprising:

- A **Language**. ModelicaML Language is based on an integration of the Modelica and SysML languages and aims at making the most of their respective strengths:
  - Modelica brings an open source acausal and declarative language for modeling multi-domains physical systems.
  - SysML brings the modeling organization of System Engineering along with requirements traceability principles.
- A **Tool**. ModelicaML Tool is based on the Eclipse framework which is definitely an asset in terms of integration capabilities because:
  - It is an open source project and based on java and XML standards.
  - It is supported by an active research community that provides features free of charge.
- A **Method**. ModelicaML Method defines a way to define properties and scenarios, to integrate models of physical processes and of control systems, to make the connections between all the models, to perform some simulations and to generate evaluation reports.

The result of the evaluation showed that:

- The **Language** brings some good features to define test scenarios, especially with the availability of activity diagrams. It also builds the premises of system requirements (or more generally system properties) traceability (possibility to link objects to requirements).
- The **Tool** is really open due to the Eclipse framework. However, the drawback of using Eclipse is that it is not user-friendly for engineers in physics (so for the people who namely develop the Modelica models of the physical processes) and that its behavior is not always predictable for people not familiar with the lower layers of Eclipse. This was definitely a point that prevented us from going as deep as we originally wanted in the evaluation of ModelicaML (i.e. from treating a real-life industrial use-case). However, we concede that this is understandable for a non-commercial tool that is built incrementally like a prototype in parallel of research activities and for which no user feedbacks are really available since the tool is rather new.
- The **Method** is in line with EDF needs concerning model-based functional validation. The added value of the method lies definitely in the connection of system properties with test scenarios and in the value binding process that permits connecting the test environment (i.e. properties models and the corresponding test scenarios) to the design environment (i.e. Modelica models of physical processes and control

systems). The method supporting the ModelicaML tool is in line with the first draft of EDF's methodology for model-based systems validation. It can even be an inspiration to improve it.

As a conclusion, even though ModelicaML "Suite" cannot be directly used in an industrial framework due to current tooling problems and lack of maturity, the method proposed is interesting and shall be merged with EDF previous work from the ITEA2 EUROSYSLIB project where a full Modelica approach for model-based system validation was investigated. As an illustration a new draft of the EDF's methodology for model-based systems validation is given as a conclusion. This work will be continued in the in-house INCOME project (for application to I&C systems validation) as well as in the upcoming ITEA2 MODRIO project (for the upstream developments concerning the tools and the languages).

It should also be interesting to keep in touch with Wladimir Schamai's work which remains still promising.

Outline

Glossary

API :	Application Programming Interface
C/C :	Control/Command
CR :	Control Room
DCS :	Distributed Control System (a computerized control system used to control the NPP)
DFE :	Diagramme Fonctionnel Elémentaire (i.e. EDF French acronym to denote a System Functional Diagram)
DFL :	Diagramme Fonctionnel Logique (i.e. EDF French acronym to denote a Logical Functional Diagram)
DSE :	Dossier Système Elémentaire (i.e. EDF French acronym to denote the folder describing a System of the NPP)
HIL :	Hardware In the Loop
HMI :	Human Machine Interface
I&C :	Instrumentation and Control
IVVQ :	Integration, Verification, Validation & Qualification
NPP :	Nuclear Power Plant
PLC :	Programmable Logic Controller
SdC :	Salle de Commande (i.e. French acronym for CR)
SED :	Distribution d'eau déminéralisée (i.e. French acronym for the demineralized water feeding system used in the SRI)
SNCC :	Système Numérique de Contrôle Commande (i.e. French acronym for DCS)
SRI :	Système de Réfrigération Intermédiaire (i.e. French acronym to denote the Intermediate Cooling Subsystem of a nuclear power plant).
ST :	Schéma-Type (a complex operator that deals with Sensor acquisition or Actuators command)
TBPM :	Time-Based Process Model
TPL :	Turn Push Light (a kind of button in the CR)
V&V :	Verification & Validation

1. Introduction

1.1. Context and goals

ModelicaML is a research project hosted by the Open Source Modelica Consortium and mainly led by Wladimir Schamaï (EADS). This tool aims at building a test environment structured and implemented in the Design and Verification / Validation activities. ModelicaML supports the "property" concept. A property is an expression that specifies a condition that must hold true at given times and places. A property is derived directly from high level requirements and has to be evaluated on a simulation model stimulated by scenarios. All these concepts are explained later in the document.

The document presents an evaluation of the ModelicaML virtual verification method and associated tool for validating systems with use of Modelica models. It corresponds to a deliverable of Work Package 6 of the ITEA2 OPENPROD project. In particular, it deals with the evaluation of the ModelicaML development made in Work Package 2 and the adequacy of this approach mixing both Modelica and SysML to validate the control systems of EDF power plants.

1.2. Structure of the document

The structure of the document is the following:

- Section 1: introduces the context and the goals of the study.
- Section 2: proposes a description of Instrumentation & Control (I&C) systems for Nuclear Power Plants (NPP), as well as current design and validation processes.
- Section 3: presents EDF needs concerning functional validation of I&C systems.
- Section 4: states the EDF's background before the OPENPROD project concerning model-based systems validation (brief summary of the main EUROSYSLIB results).
- Section 5: presents the ModelicaML method and tool.
- Section 6: gives a conclusion to the evaluation.

1.3. References

[DR1] ENSISP110148 - *Note d'opportunité de rénovation des simulateurs d'étude pour l'ingénierie du parc en exploitation*

[DR2] IEC 61513 - *Nuclear power plants - Instrumentation and control for systems important to safety - General requirements for systems*

[DR3] IEC 60880 - *Nuclear power plants - Instrumentation and control systems important to safety - Software aspects for computer-based systems performing category A functions*

[DR4] AIEA NS-G1.3 - *Instrumentation and Control Systems Important to Safety in Nuclear Power Plants - Safety Guide*

[DR5] NLF-F DC 634 - *SPPA-S2000 (S5) Emulation Credibility file (Written by AREVA)*

[DR6] IEEE 610.12-1990 - *Standard Glossary of Software Engineering Terminology*

[DR7] Cours Test de Logiciels - Bruno Legeard: Laboratoire d'Informatique de l'Université de Franche-Comté. INRIA & CNRS

[DR8] H-P1C-2011-00913-EN - *EUROSYSLIB Project - sWP7.1 - Properties Modeling*, Jardin A., Nguyen T., Ruel N., August 2011.

[DR9] IEC 60964 - *Nuclear power plants – Design for control room of nuclear power plants*

[DR10] ModelicaML web page: https://www.openmodelica.org/index.php/developer/tools/134 (where to find ModelicaML environment installer and tutorial documents written by Wladimir Schamaï)

[DR11] Jardin A., Bouskela D., Nguyen T., Ruel N., Thomas E., Chastanet L., Schoenig R., Loembé S., *Modelling of System Properties in a Modelica Framework*, in proceedings of the 8th International Modelica Conference, March 20th-22nd, Dresden, Germany, 2011.,

[DR12] *Full Project Proposal of the OPENPROD project*, ITEA2 report, 2009.

2. I&C systems of Nuclear Power Plants

Before presenting EDF requirements concerning the use of physical models for validating NPP I&C systems, let's describe at first what are these I&C systems and the current engineering practices in the nuclear domain.

2.1. Purpose of NPP I&C systems

The goals of the I&C systems of Nuclear Power Plants (NPP) are to:

- Drive the plant,
- Protect the plant,
- Protect people working in the plant,
- Protect the environment.

The overall I&C system is part of a set of means developed in order to:

- Ensure the correct operation of the nuclear reactor,
- Prevent incidents and accidents,
- Limit the consequences of potential incidents and accidents.

2.2. I&C system architecture and technologies

I&C system in Nuclear Power Plants is structured in 4 levels:

- Level 0 is composed of field equipments and constitutes the interface with the physical process (sensors, actuators and electrical power supply);
- Level 1 performs regulation and protection automated functions;
- Level 2 provides a system interface to operators and processes data going to Level 3;
- Level 3 gathers plant specific data that are exchanged with Level 2 or Level 1 and with national supervision systems (generation of reports, curves analysis ...).

For level 0, several kinds of instrumentation systems are installed in the plant:

- Nuclear instrumentation (e.g. neutron flux internal and external measurement, rod positioning measurement, …);
- Thermo hydraulic instrumentation (e.g. temperatures, pressures, flows, levels, …);
- Other process instrumentation (e.g. Boolean on valve openings/alarms, positions, speeds, frequencies…).

For level 1, the controllers can be of different kinds:

- Logic controllers: electromagnetic or programmable;
- Regulation controllers: analog or numeric;
- Protection controllers: electromagnetic, static or programmable.

For level 2, several supervision means are available:

- For the sake of information and monitoring:
  - Conventional means: indicator lights, alarm indicators, analog indicators and recorders;
  - Computer-based means: screens, printers, indicators and recorders;
- For the sake of control:
  - Conventional means: TPL (Turn, Push, Light), push buttons, commuters, lock keys;
  - Computer-based means: keyboards, trackballs, touch screens.

For level 3, report generation and analysis as well as connection with national supervision are performed with computer-based components operating on mass market operating systems such as MS Windows.

All these levels are interconnected through:

- Hardwired connections (no specific modulation or communication protocol) - this is the preferred connection type for safety functions and protection controllers.
- Multiplexed connections (data is encoded, transported by a communication protocol and supported by a communication bus).

As an illustration, Figure 1 presents the Instrumentation and Control structure for a 1300 MW nuclear power plant:

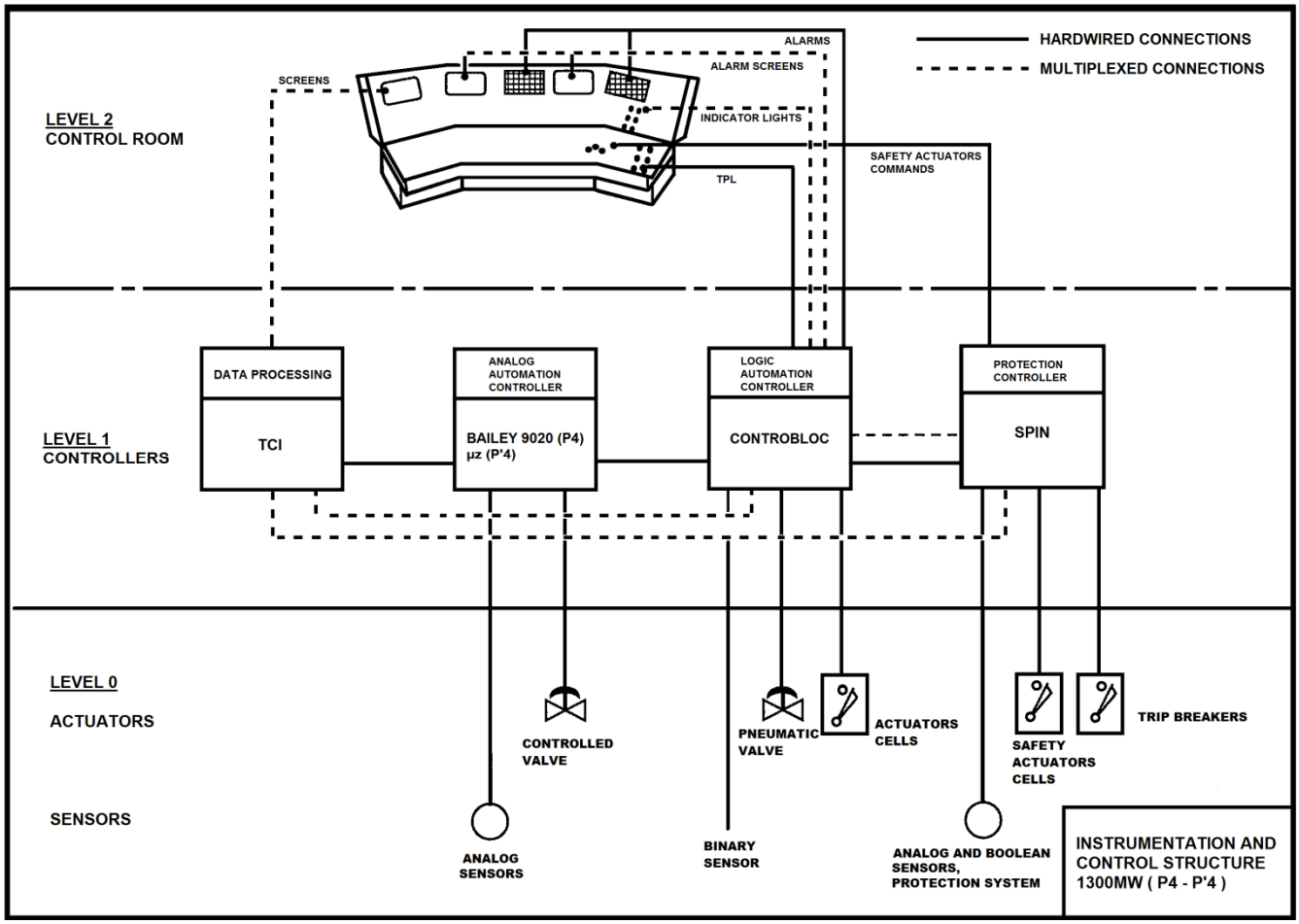

Figure 1 : Illustration of the I&C System of a 1300 MW nuclear power plant

The architecture of the NPP I&C system derives from this structure and implements several design criteria's like: defense in depth, reliability, independence, failure modes, common cause failure, performance, qualification, testability, maintainability …

2.3. Current development lifecycle of automation and control systems for NPP

Hereafter is presented an overview of the current development process for NPP's I&C systems that is ruled and described in the IEC 61513:

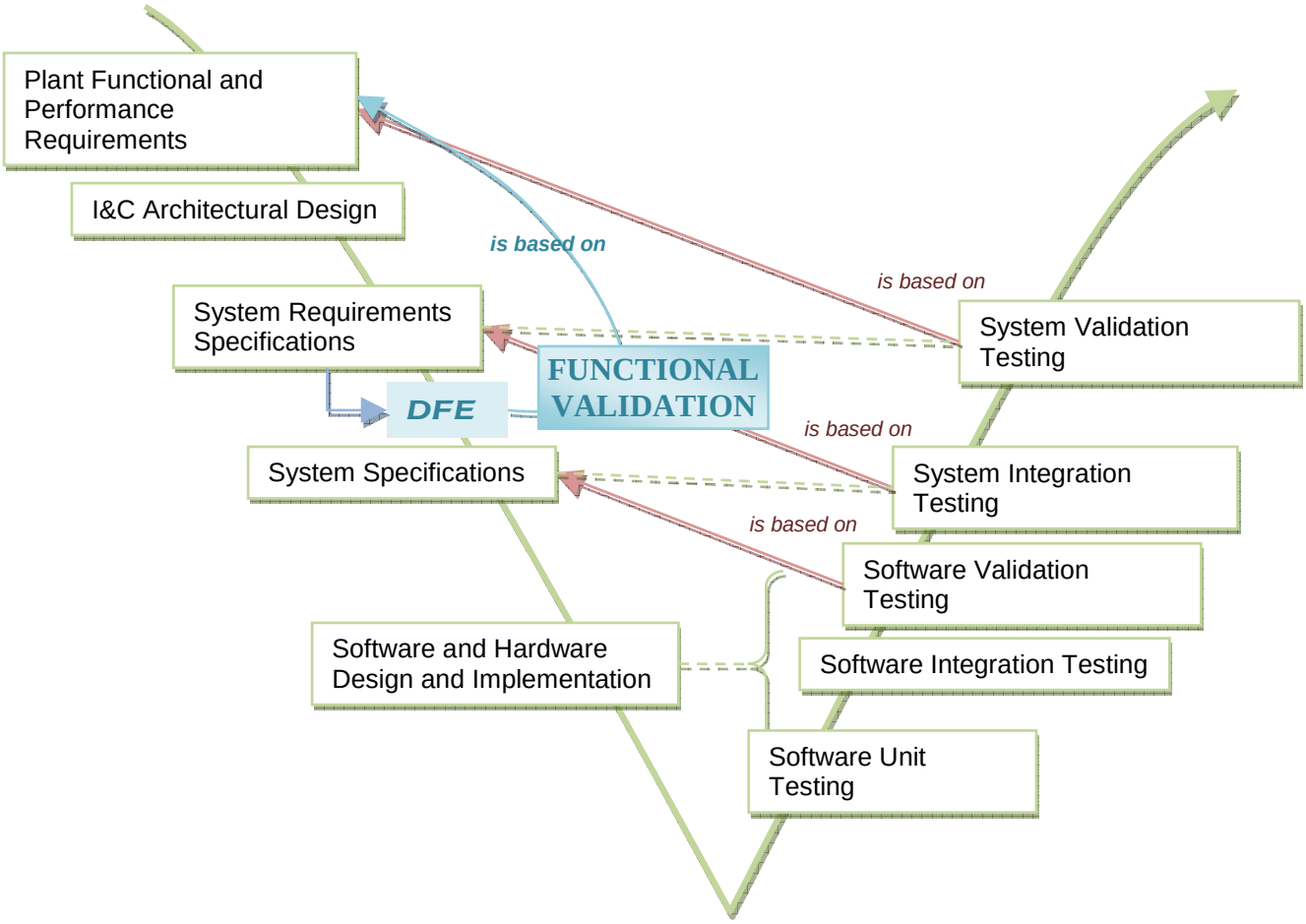

Figure 2 : I&C systems development cycle

Testing activities deal with validation of design or implementation activities that are positioned at the same level. Testing activities are then based on the description given by the upper level of specification.

2.4. Use of Functional Diagrams (or DFE)

DFEs (for Elementary Functional Diagrams) are a graphical way of formalizing functional specifications. These diagrams have a semantic and can be simulated in tools like Simulink or Dymola. DFEs construction is an important step of Elementary System design and has to be validated and verified with care before being implemented.

DFEs support functional design of I&C systems for EDF nuclear power plants. DFEs are data flow diagrams based on components libraries. Components can be either simple function blocks like AND, OR, THRESHOLD … or complex function blocks also called "Schémas Type" in French which are presented after in the document (see Section 2.5).

Figure 3 illustrates an example of a DFE:

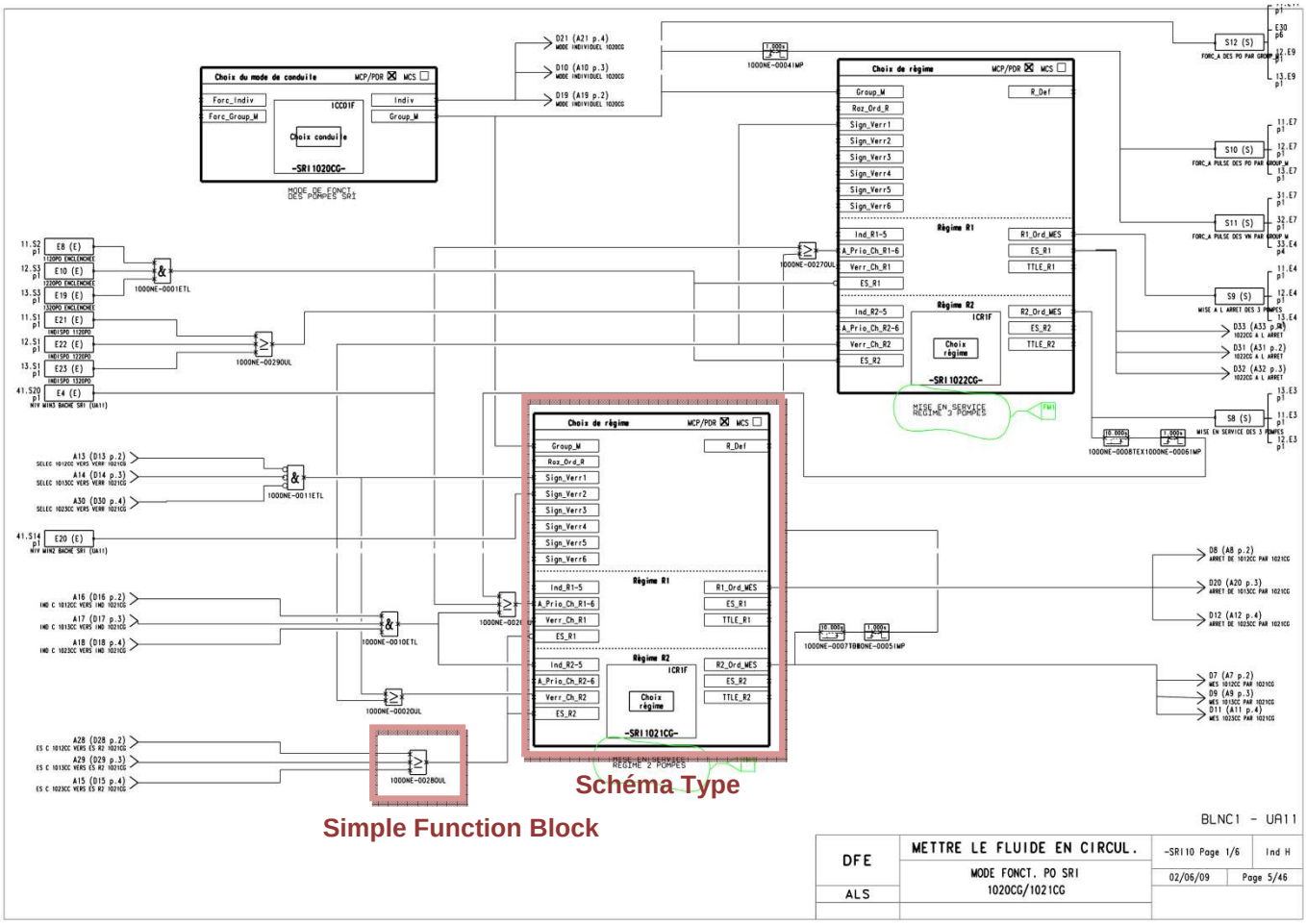

Figure 3 : Example of a Functional Diagram (DFE) for EDF I&C system

Inputs are represented on the left, outputs on the right.

Functional Diagrams (or DFE according to EDF terminology) are the main input documents for Level 1 controllers' specification. Functional Diagrams are not only used in the design phase but also during the operation of the plant units: this is why functional diagrams are based on a Function Block graphical language that should be easily readable by a non expert or a NPP operator.

Complex function blocks (or “Schéma Type” according to the EDF terminology) implement a behavior that potentially covers several architecture levels as it is further explained in the next section.

2.5. Use of Complex Function Blocks (or “Schéma Type”)

Complex Function Blocks (or "Schéma Type") are structured in this way:

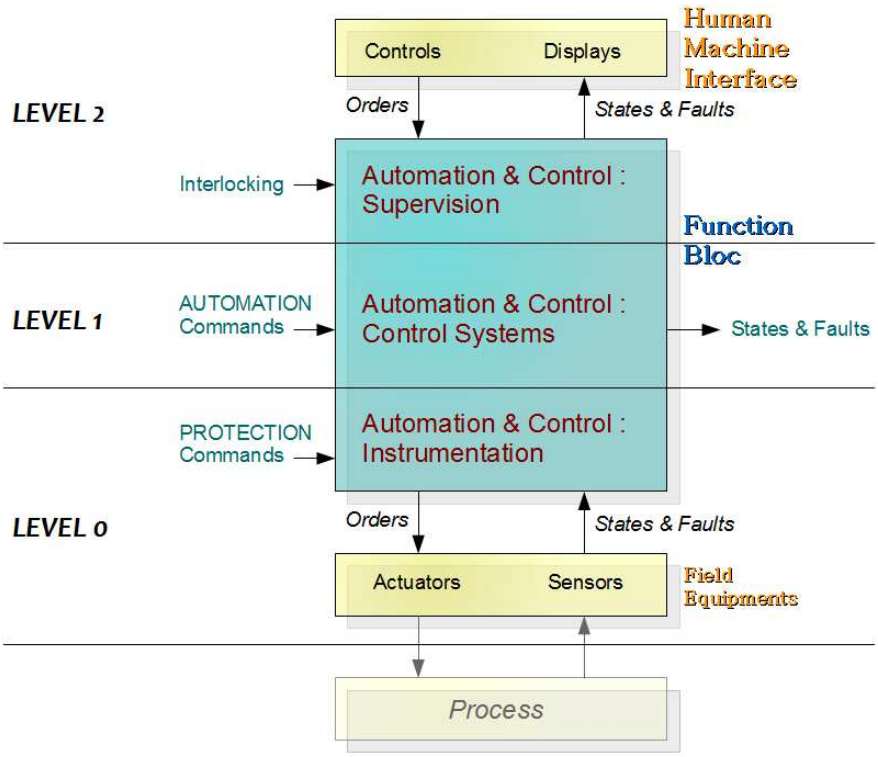

Figure 4 : Structure of a Complex Function Block (or “Schéma Type”)

This kind of Function Block is a complex object that embeds instrumentation, control system and supervision behavior. Using such a Function Block in a Functional Diagram defines automatically control system interfaces with supervision and instrumentation.

Example: motor pump command, resistive temperature detector (RTD), sensor …

3. EDF needs concerning the functional validation of I&C systems

IEC 61513 gives a definition for functional validation as quoted below:

> **3.24 Functional Validation**
>
> Verification of the correctness of the application functions specifications versus the first plant functional and performance requirements. It is complementary to the system validation that verifies the compliance of the system with the functions specification

This activity is positioned on the Figure 2 development lifecycle among other design and V&V activities.

EDF aims at formalizing and reinforcing its functional validation activities by using modeling and simulation tools. Indeed, functional validation of specifications could help detect defaults earlier and prevent their apparition in later steps of the design. This is very important considering the costs of late default detection in processes such as in nuclear industry and more specifically Nuclear I&C.

EDF is currently elaborating a methodology to functionally validate I&C systems of its power plants and this with the following constraints:

- The functional validation methodology shall be completely integrated to EDF I&C design activities. It shall make the most of every piece of specification or design available. It could be necessary to adapt a bit the design process so as to provide the right inputs to the V&V activities.
- The functional validation methodology shall be as independent as possible from software providers so as to guarantee durability. The strategy is then to base the methodology on generic concepts, open languages and open source tools as much as possible.
- The functional validation methodology shall be easily implementable and adaptable to different project cases we encounter at EDF: new builds and refurbishments.

For EDF projects, the role of I&C systems becomes indeed more and more important: they must satisfy numerous objectives in terms of safety, dependability and performance. Due to ever increasing safety constraints and environmental rules, the following question is hence of prime interest: will the plant (their physical parts together with their I&C systems), as specified and designed, guarantee the properties expected?
Currently, power plants models are already used to study several physical phenomena on different time scales. The problem of such a verification is that for the EDF nuclear power plants, the required properties can be found mainly in a textual manner and in paper documents intended to the engineering entities. A first observation can thus be acknowledged: there is a real need to formalize and capitalize on these different properties in order to facilitate the transmission of knowledge on the power plant requirements (i.e. overall goals, components constraints and modeling assumptions), to keep track of the requirements evolutions and design choices, and to remove the sources of ambiguity that could lead to misunderstandings.
Besides, although some studies already include I&C alarms that monitor key properties, many simulations rely on these alarms to determine whether the plant system is robust during transients. However, one can argue that it is inappropriate to rely on I&C to assess the robustness of a plant system, as an error in I&C, like a missing alarm or an insufficient set of sensors, could lead to a wrong conclusion. One needs to ensure that the monitoring by the I&C does reveal all properties violations. Moreover, correctness of the I&C implementation with respect to its specifications may not be sufficient, as there is a potential for the specifications to be incomplete or worse incorrect. A more appropriate approach to validate the physical system as well as its I&C system is to define properties as close to the physical system as possible. For example, whereas the I&C system computes the neutron flow rate from a limited number of measurements, the system properties to be monitored concern the flow rate itself. This latter approach has also the benefit that, if I&C systems add filtering to the raw values acquired from sensors, on the contrary, system properties should not make use of such filters (Figure 5). Also, properties modeling is simpler than I&C modeling.
All of these reasons explain why EDF intend to use the notion of system properties in its V&V activities. In particular, EDF's objective is to model properties separately from I&C, as shown in Figure 5, so as properties

checking bypasses the I&C system, and therefore can be used to check the I&C system itself, either as a black box (no specific properties on the I&C system are involved in the checking) or as a white box (specific properties on the I&C system are involved in the checking).

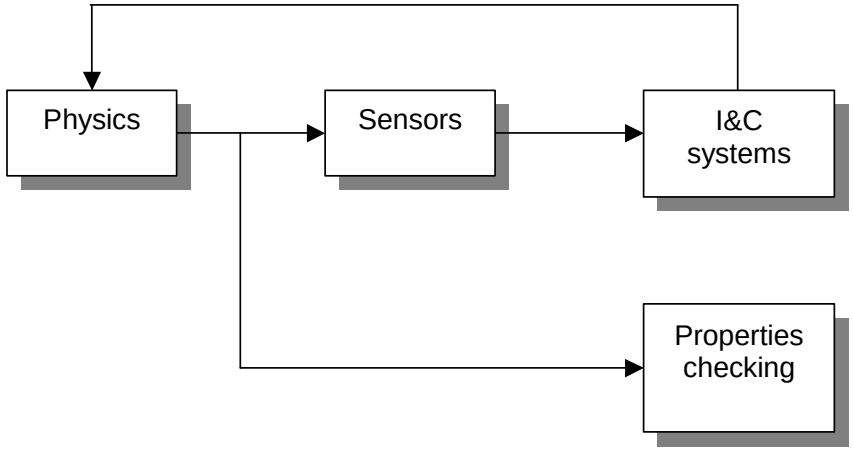

Figure 5: Location of the system properties checking (bypassing the I&C systems)

In other words, by formalizing the required properties (and most particularly the safety properties) in an appropriate language, it is expected to:

- enhance the demonstration of safety by validating I&C functional requirements specifications through joint modeling and simulation of physical and I&C subsystems;
- improve engineering processes with explicit and unambiguous specification of assumptions regarding the environment of the system or the models used for the studies, of the properties required on the system behavior, of the designer's assertions regarding the system internal behavior, of the validity limits of a model, etc;
- improve documentation on operating domains and on functional constraints of systems and components in order to ease capitalization and transmission of knowledge (e.g. the extension of power plant lifetime to 60 years or more needs to provide appropriate information for future upgrades 30 or 40 years from now). In shorter terms, there is a need to provide appropriate information to the users of a components library;
- keep track of the evolutions of the system properties (e.g. due to design improvements, to changes in operational expectations, to model refinements, …);
- check the consistency and the completeness of the requirements (e.g. by formal proof and consistency checks) and verify the conformance of the system to the expectations (e.g. by simulation);
- support advanced modeling approaches like model switching (e.g. simulation of a system entering a dysfunctional mode with a model different than the one representing nominal conditions).

In consequence, to suit the EDF needs, a verification tool should be able to support the following main activities:

- Identify requirements in "plant functional and performance requirements" documents;
- Translate these textual requirements into properties that are formalized and executable;
- Dispose of executable I&C diagrams;
- Dispose of nuclear process simulation models;
- Define test scenarios that would put the I&C and the process models in typical operation conditions (e.g. conditions which are most dimensioning) and call the corresponding properties model in order to verify that these properties are not violated all along the test scenario that is simulated.
- Provide a test report stating and proving that the test passed or giving sufficient data log to investigate why the test went wrong.

Test scenarios and properties defined at this stage for functional validation purpose should also be reused later, for system validation.

Finally, process models are to be part of the simulation loop as well in order to recreate same conditions than for functional validation. Consequently, in order to perform system validation tests, with real I&C equipments, EDF will need to dispose of real-time Hardware In the Loop (HIL) test means.

Previous work led at EDF's has permitted to:

- Choose the Modelica language for process modeling and simulation activities (at a 0D/1D level). A big experience has indeed been acquired on the language. EDF has in particular developed its own thermo-hydraulic library, called ThermoSysPro, which is open source and which is used to model different kinds of power plants (nuclear, thermal, combined cycle, solar, …).
- Acquire a big experience on Dymola which is the tool the most often chosen to build in-house Modelica models since it appears as the most mature from the industrial point of view and is up to now considered as a reference by the Modelica community.
- Pave the way to a model-based test methodology especially with the definition of properties as a result of the ITEA2 EUROSYSLIB project. This project showed that Modelica language gets some good characteristics in order to express properties (e.g. like being a declarative language). Results of EUROSYSLIB are developed in the next chapter.

EDF has already performed some work on properties modeling. OPENPROD project is the opportunity to go further with the investigation of ModelicaML Language and tool capabilities. In the following chapter, EDF background in properties modeling is presented.

4. EDF background on model-based system validation with the use of properties models (EUROSYSLIB results)

This chapter presents the main results of EUROSYSLIB project. These results can be considered as the EDF state of the art of EDF good practices in terms of properties usage in a test environment. The descriptions that come hereafter directly come from [DR8].

4.1. Use and interest of properties for model-based system validation

A system property is an expression that specifies a condition that must hold true by the system at given times and places. It results in a Boolean value stating whether the property is satisfied or not.

Formalizing properties could be of good help for supporting I&C Systems functional validation. Indeed, the functional specification of I&C systems is usually close to its implementation. There is thus a risk that the bench of test scenarios does not offer a sufficient coverage because of a lack of independence between specifications and implementation. Basing testing activities on properties that derive directly from the upper level of specifications could help improving test coverage. Properties assessment could be of different kinds and serve different purposes:

- Static evaluation (to check consistency and/or completeness of the specifications …): the verification is based on the formal descriptions of the specifications (e.g. for a model where two properties should be satisfied, a first check should ensure that there is no contradiction between these two properties, and this before any simulation run).
- Dynamic evaluation (to check the satisfaction of functional, dysfunctional, performance constraints …): the verification is based on a co-simulation of the properties model with the behavioral of the physical system.

Note: More often, properties are understood as modeling specifications and/or components constraints but they could also be used to define the validity domain (or the modeling assumptions) of a behavior model.

4.2. Formal expression of a property

4.2.1. Locators

The formal expression of a property can be expressed in the following manner: [**Where**] [**When**] [**What**]

- The "***where expression***" is a space locator that specifies which part of the system is concerned by the property. This part can be a set of components (e.g. all the pumps of the system), a specific subsystem or an individual component (e.g. the first pump or the cooling subsystem), a geometric region of a component (e.g. a specific segment of a pipe), and so on.
- The "***when expression***" is a time locator that indicates at which instants the property must be satisfied. This locator can be a time instant (e.g. when a pump starts), a time period (e.g. as long as the pump is on), a sliding time interval (e.g. for no more than 3 minutes over any period of 2 hours), and so on.
- The "***what expression***" refers to the condition the system must guarantee. Due to the variety of properties, this condition can be of multiple kinds and involve physical variables as well as events or states of specific subsystems or components (e.g. the inlet pressure must be higher than a minimum value, Pump1 must never start more than three times per sliding time interval), and so on.

4.2.2. Attributes of a property

It has been shown in the previous section that a property may entirely be defined by the association of three kinds of attributes, namely a space locator, a time locator and a condition to be satisfied. However, each kind of attributes involves several kinds of objects such as instances of model, geometric data, physical variables, states and events. The aim of the following section is to recall the definition of these objects to better understand how they can be implemented in Modelica and/or ModelicaML.

- Instance of a model: An instance of a model refers to a specific object that obeys to the same rules or equations than the corresponding model. The name of the instance is in particular useful when the same submodel is used several times in the same model.
- Geometric data: A geometric data defines a particular region of a system, a subsystem or a component, according to specific space units.
- Physical variable: Physical variables and their evolutions are continuous variables used to represent the physical quantities of a system, a subsystem or a component.
- System state: A system state is a discrete variable that characterizes an aspect of a system, a subsystem or a component. It can take its values only within a finite set of values.
- Event: An event is an object that is generated at a given time to signal the occurrence of a fact. It carries two pieces of information:
    - The date of the event indicates when the fact has occurred;
    - While the class of the event defines what has happened.

  Also, an event may be characterized by additional information such as a probability distribution, a frequency …

4.2.3. Operators

From the previous section, several kinds of objects have been defined such as instances of model, geometric data, physical variables, states and events. Now that these notions have been introduced, they can be combined using operators to build property attributes.

An operator is a function that constructs and outputs one or several objects, given one or several objects as inputs. Inputs and outputs may be of the same type or of different types (*e.g.* physical variables, states, events …). An operator may in particular be used to build a property's attribute (*i.e.* a space locator, a time locator or a condition):

- When a simple observation of the variables available in the behavioral model is not sufficient to describe what the system must guarantee;
- When it is easier to express it as a function or a combination of other attributes.

From the EUROSYSLIB analysis of EDF and Dassault Aviation needs, some operators were identified to be useful such as:

- Arithmetical operators and usual functions:
    - +, -, *,, cosine, sine, logarithm, absolute value, …
- Logical operators:
    - $\cap$ (and), $\cup$ (or), $\neg$ (not), …
- Set operators:
    - all, $\in$, $\notin$ (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, after, before, always, never, delay between two events, duration of a time period, count of the number of events, events synchronization,…
- Dedicated operators:
  - >, <, $\geq$, $\leq$ (thresholds), $\subset$, $\not\subset$ (inclusion), Δ (ramp), A (accumulation), $freq(\ )$ (frequency), …

4.3. Requirements to model properties

A property has to be:

P.1. in ***Interaction*** with the behavioral model of the system

The assessment of a property is based on the evolution of the system behavior and of the system environment.

P.2. Transparent for the dynamic evolution of the system

Properties assessment must be transparent and invisible to the modeling and the simulation of the system physical behavior:

- during the modeling phase, properties must be considered as intrinsic characteristics that the system, subsystems and components must satisfy. Thus they must not be used to define the physical behavior of the system;
- during the simulation phase, the evaluation of a property must not affect neither the dynamics of the system, subsystems and components nor the parameter values chosen to depict the physical behavior of the system.

Reciprocally, the choices made to model or to simulate the system physical behavior should not influence the modeling of properties[1].

P.3. Observable from the behavioral model by not implying a too low level of details

Properties may be specified at different levels (e.g. a property may concern a system, a subsystem or a component, may refer to functional aspects or detailed designs) but the variables used by the properties model must be computable from the variables available in the behavioral model.

P.4. Readable for the sake of documentation and transmission of knowledge

Properties must be clear. By reading the properties model, one should be able to understand without error the meaning of each property, and determine on what, where and when it applies.

P.5. Understandable to ease the interpretation of its potential non-assessment

Dynamics of properties must be understandable. By seeing the simulation results of the properties model, one should be able to explain why a property failed.

The modeling of properties must then be in accordance with these different principles and an adequate conceptual model has to be established in particular to guarantee the transparency of the properties model with respect to the behavioral model.

[1] This principle is also true when properties are used to define the validity domain of a behavioral model. In this case, the definition of a property should not be modified by the equations of the model, and vice versa.

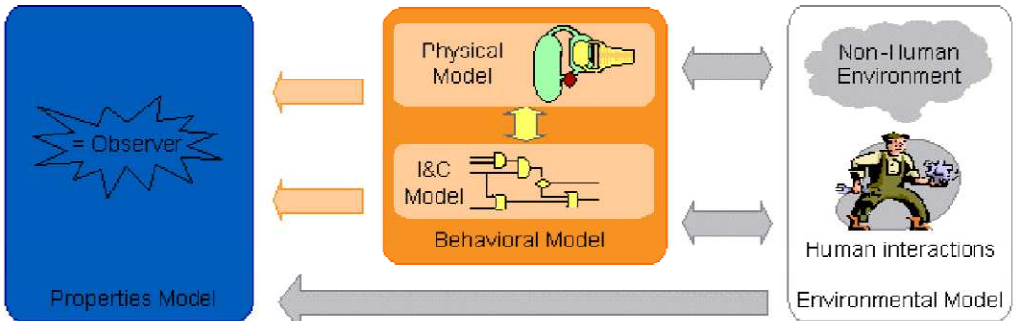

Figure 6 : Conceptual model for modeling the system behavior, the evolution of its environment and the properties the system must guarantee

Such a conceptual model corresponds to a model organized in three different parts (Figure 6):

- the environmental model where the characteristics and the evolution of the system environment are specified (including the human interactions). This part may in particular be used to set the inputs of the simulation and so to specify scenarios depending on the type of study under consideration (e.g. introduction of some component failures, simulation of a sequence of operator interventions, …);
- 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 system;
- the properties model where the expected services of the system and the validity domain of the behavioral model are specified.

In order to ensure the fact that the properties model must be only an observer of the behavioral model, the three parts of this conceptual model must communicate such that:

- the properties model and the behavioral model may access the data provided by the environmental model (the properties as well as the behavior of the system may actually depend on the evolution of the system environment)
- the properties model may access the data provided by the behavioral model in order to evaluate the dynamic evolution of the physical process and its I&C system, 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 system under consideration (Figure 6(a));
- The different states of the system and its environment (Figure 6(b));
- A combination of the system architecture and of the different states (Figure 6(c)).

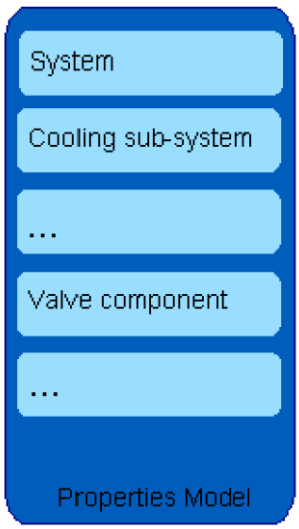

Figure 6(a): Hierarchy according to the system architecture

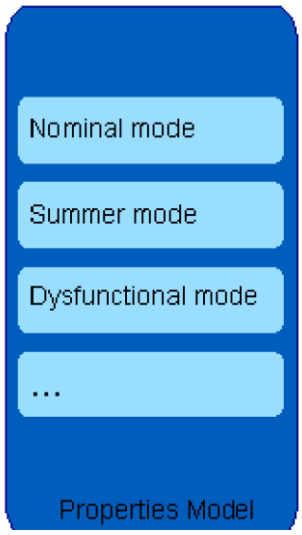

Figure 6(b): Hierarchy according to the different operating modes

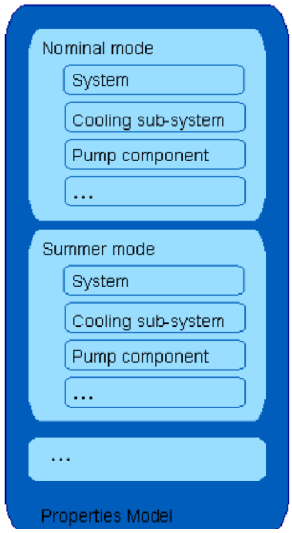

Figure 6(c): Hierarchy according to the different operating modes and the system architecture

Figure 7 : Different ways a properties model can be organized

A hierarchy based on the system architecture may be useful in particular when the architecture of the system is modified and the modeler has then to remove or add some properties related to specific subsystems or components.

A hierarchy based on the states of the system and of its environment may add further information on how the system must behave (the description of these states gives in general a better insight into the different operating modes). Such a hierarchy is made even more natural by the fact that properties are often closely related to these states.

As a consequence, the conceptual model given in Figure 6 corresponds in fact to a simplified view point. In practice, since a properties model can be built per each operating mode, several properties models can be associated with the same behavioral model. A more advanced conceptual model is needed to handle such a situation. One possible solution is to add a statechart model where the different states of the system and of its subsystems and components are described. The main idea is that the statechart model is viewed as a supervisor: it may access the data of the behavioral model, decide in which state the system operates and select the appropriate properties model.

Finally, in the same spirit, it might also be useful to associate several behavioral models with the same system (Figure 8). For instance, if the objective is to predict the physical behavior of the system when a fault occurs and to verify that the system remains afterwards in the safe domain as specified by the properties model, it might be helpful to switch, during the simulation, from a model describing the nominal behavior to another model describing the dysfunctional behavior of the system.

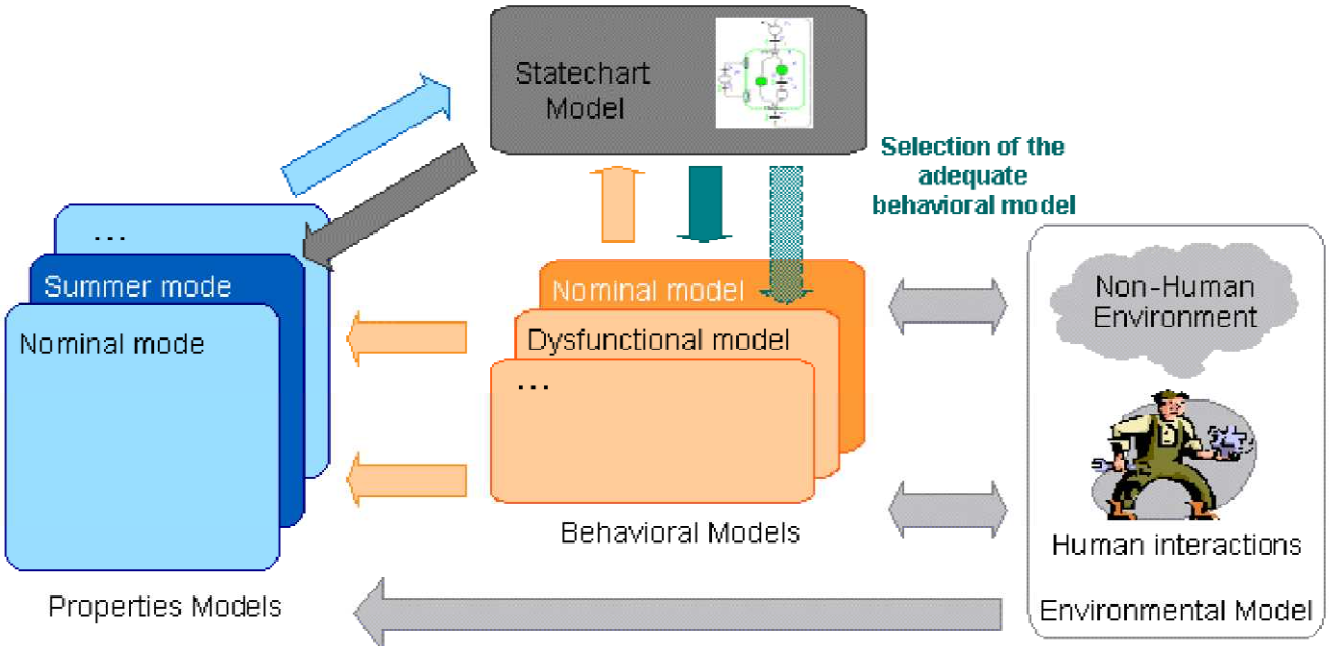

Figure 8 : Advanced conceptual model for properties model and behavioral model switching

4.4. Requirements to check, visualize and analyze a property

Initially, both models (i.e. properties and behavioral models) are expressed separately because they require different skills and have different lifecycles. Both these models should also be designed separately for safety purpose. The objective here is then to couple these models in order to check that the properties are satisfied by the system modeled.

Two checking approaches may be considered: one based on formal proof, and one based on simulation.

Check by formal proof can perform static analyses in order to verify

- the compatibility of the specified properties (e.g. to ascertain that they do not define an empty operating domain);
- the consistency between the properties and the behavioral model (e.g. to check that the behavioral equations are mathematically compatible with the properties).

Complementarily, checking by simulation can perform dynamic analyses in order to verify that the properties are satisfied all along a given scenario (e.g. during a scenario featuring sequential changes of operating modes).

As a consequence, even if several conceptual models have been suggested in the previous section to satisfy some users' requirements in terms of modeling, further requirements regarding the checking and the visualization of results must also be added.

In order to enable properties checking by formal proof, the language used to describe properties and behavioral models should allow a formal description.

In order to enable properties checking by simulation, the software tool used should allow the definition of scenarios with occurrence of dysfunctional modes, faults injection, etc.

Besides, in order to guarantee the understandability of the properties, this tool should also offer diagnostics like:

- generation of alarm when a property is violated;
- change of component's visual aspect;
- edition of a log file;
- selection of different classes of properties;
- …

5. ModelicaML for model-based system validation (OPENPROD evaluation)

The objective of the document is to study to what extent system properties can be modeled and then associated with an existing Modelica behavioral model for the needs of the Verification and Validation (V&V) process.

Starting from the agreement that UML/SysML is more specialized into the properties capture and that Modelica enables a refined description of the physical behaviors, two possible approaches for managing such a study are the following:

1. "Either one tries to strengthen the UML/SysML capabilities by embedding the force of Modelica to model physical behaviors,
2. Or one tries to enable the modeling of properties directly into the Modelica language."

The EUROSYSLIB project led the evaluation of the second approach which is recalled shortly in Section 4.

The following sections will detail the first approach with ModelicaML and compare it against what was performed with Dymola and against the expectations regarding a property edition language.

5.1. ModelicaML tool framework

First of all, here is a small introduction of the ModelicaML environment [DR10].

ModelicaML is a modeling language that merges the strength of SysML and Modelica languages. Modelica is a declarative language that permits acausal simulation of multi physical models. SysML proposes capabilities for modeling systems under several points of view. SysML provides a structured approach for model-based system engineering. ModelicaML purpose is to integrate Modelica into SysML.

The ModelicaML language is:

- Put forward with a test methodology based on the evaluation of system properties;
- Implemented on an Eclipse-based framework.

The Eclipse-based framework that supports ModelicaML is built around the following modules:

- Eclipse Modeling Tools: provides the Eclipse environment basis for modeling projects.
- Papyrus: UML and SysML modeling environment based on Eclipse and provided by the CEA LISE.
- MoDisco: is the model explorer customization. In other words, it permits creating a modeling interface (graphical objects, palette to access libraries …) connected to the meta-models.
- Xtext: permits ModelicaML action code editing (e.g. syntax coloring based on the lexical structure) valid code completions and static analysis and validation.
- Acceleo 2.8: for Modelica code generation from the ModelicaML models.
- Modelica Development Tool: used for viewing and editing Modelica code within the Eclipse environment and for connecting the ModelicaML tool to the OpenModelica Compiler (OMC) for model validation and simulation.

5.2. Associated test methodology for V&V activities

ModelicaML is a language which implementation in Eclipse framework is performed such a way that it suggests a methodology. This methodology is presented here.

The models of the process, of the control system as well as the properties and the test scenarios models can be designed independently from each other. All these models could be designed directly in the ModelicaML Eclipse environment since it offers a Modelica editor. Some Modelica libraries are also available and additional libraries could be loaded such as the EDF thermo-hydraulic library, called ThermoSysPro, provided these libraries are supported by OpenModelica.

Hereafter is presented the structure of a ModelicaML project:

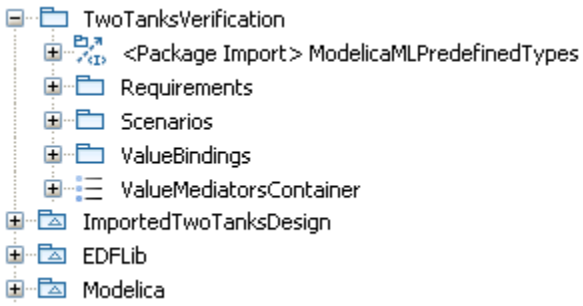

Figure 9 : Project context view

The modeling of physical processes and control systems is however much easier in the Dymola environment due to better graphical design features. Nevertheless properties and scenarios models could benefit of ModelicaML capabilities and be designed in this environment.

Here is a representation of a simple process model:

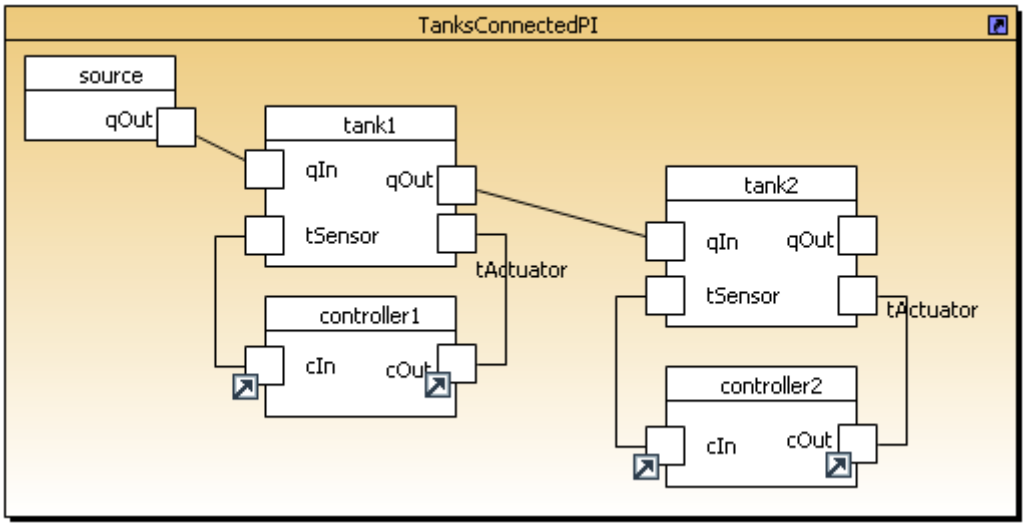

Figure 10 : Modeling in the ModelicaML tool

As we can see on the figure above, this is potentially possible to model every kind of system but:

- The scheme is not expressive: only boxes are represented and there is no component/graphical icons to ease the reading of the scheme.
- Modeling is more complicated than a traditional Modelica tool since every component interface node (e.g. *tActuator*) has to be created manually each time even though the component (e.g. *tank*) is part of the

generic library and defined to dispose of the interface. In other words, every object of the scheme have to be dragged and dropped manually

In parallel, ModelicaML provides several features to define properties and test scenarios:

- Properties can be defined in two different ways:
  - With the ModelicaML Language facilities: properties can then be coded in Modelica (Figure 11).
  - With statechart diagrams: edition of properties based on states monitoring (Figure 12).

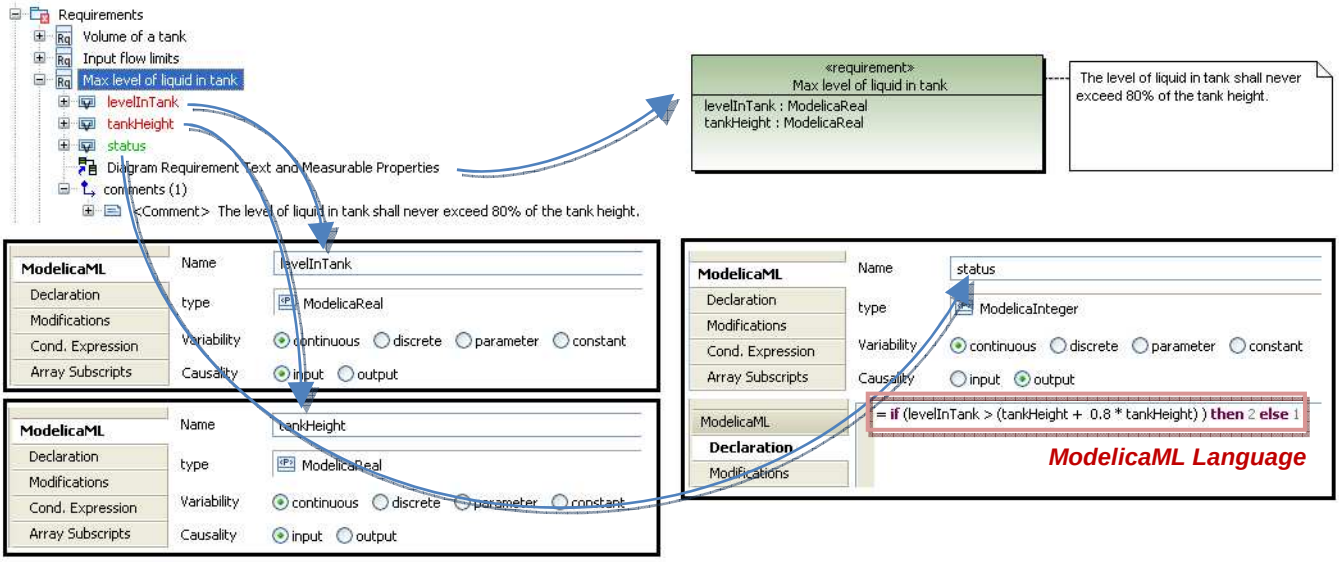

Figure 11: Properties modeling with ModelicaML

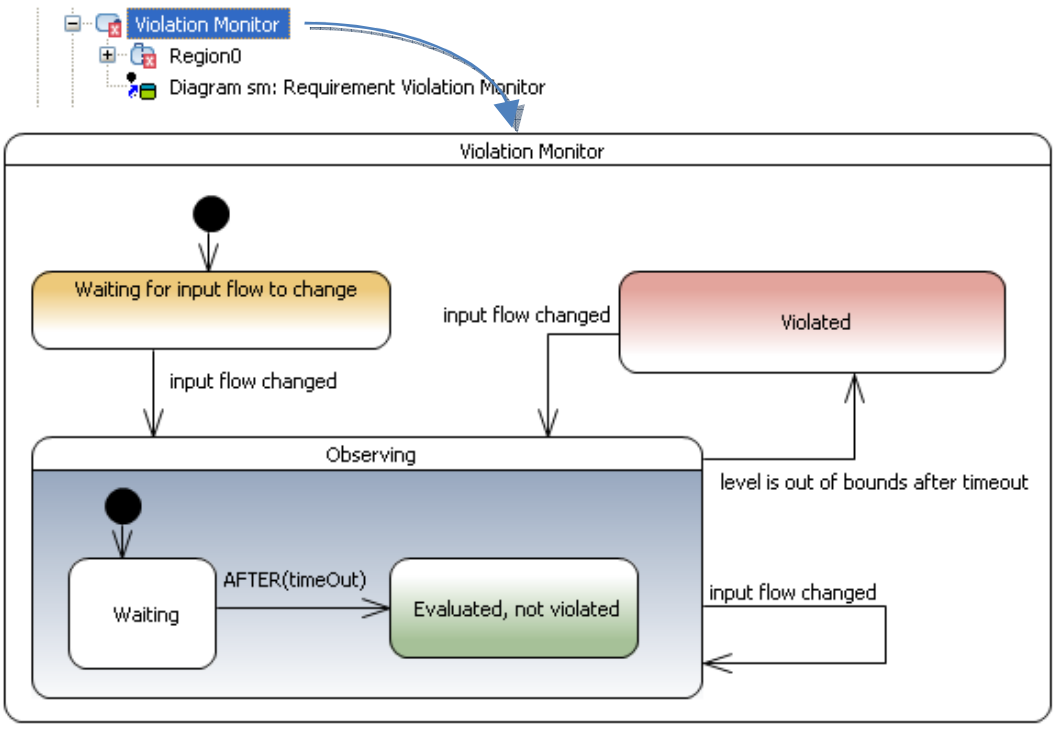

Figure 12 : Properties modeling with statechart diagrams

Here, the property “the level of liquid in thank should never exceed 80% of the tank height” is described. It uses the “tank level” and the “tank height” as inputs. The status gives the result of the property evaluation. The status is elaborated thanks to the execution of an expression written in Modelica language: this is the property.

The violation monitor looks like a state diagram in which states are the state on the property evaluation and transitions carry conditions for the property evaluation.

- Test Scenarios can be defined in two different ways:
  - With the ModelicaML Language facilities: algorithms (basic operators) may be used such as: if then else, >,<, == …(Figure 13)
  - With statechart diagrams: edition of test scenarios based on a sequence of inputs for the system under test separated by transitions activated by system states changes or time passing (Figure 10).

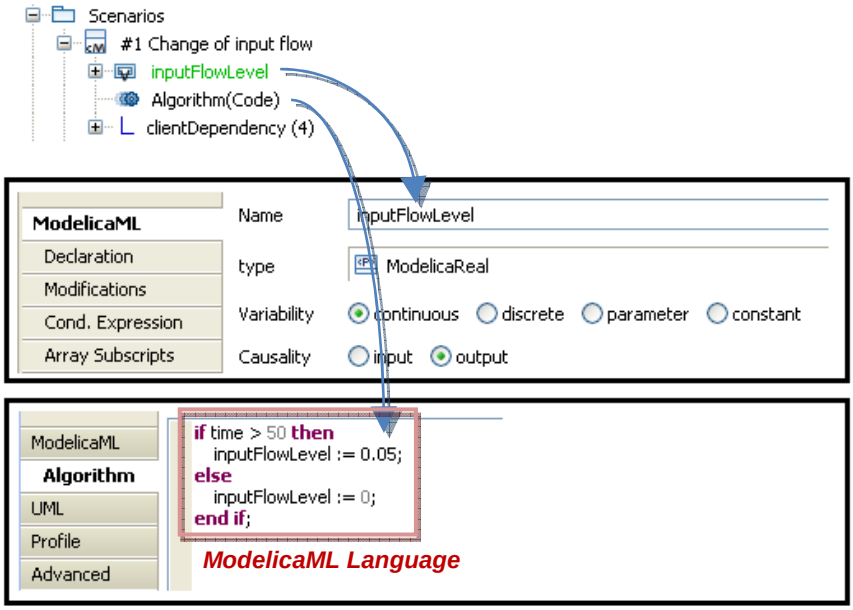

Figure 13 : Modeling of a test scenario with ModelicaML

Another set of models may be created such as states supervisors. These models are built on the basis of the process and control system in order to supervise the states of the overall system. Expression of properties and test scenarios may need states monitoring. For example, the state of a component can be monitored by a property or the state of the system can be used as a condition so as to trigger a test scenario.

The states supervisor models could be split into two categories:

- Single components state supervisor: the state definition is tightly linked with the component definition and its parameters. It can be directly associated with the library component, preferably in the environment used to model the process and the control system such as Dymola. Such a supervisor model is directly linked to the component.
- Global system state supervisor: depends on the system function and specification. As it derives partially from the specification, it could be initiated from a test point of view, preferably, in the ModelicaML design environment. Indeed, the design of the system state supervisor is linked to Verification and Validation activities and will depend on the test objectives.

While system and test models are to be designed independently, the states supervisor models shall ensure the consistency of all models in terms of connection. This can be performed through an iterative design process between the different engineering teams of the system development lifecycle.

The ModelicaML environment enables the importation of existing Modelica models. The "**.mo*" files of the process and control models are parsed in order to identify the variables and the parameters the test models will interact with. In case these models are built on a specific library, such as ThermoSysPro, it is possible to import this library in ModelicaML tool.

In the following example, one single global Modelica file has been generated by Dymola. This file contains the model of the SRI system (including the representation of its physical process and its I&C system), its associated properties model as well as the EDFLib and Modelica Standard Library. The user can choose to synchronize independently every part of the imported package. This is interesting in case one need to deal with different evolutions of the process or control models.

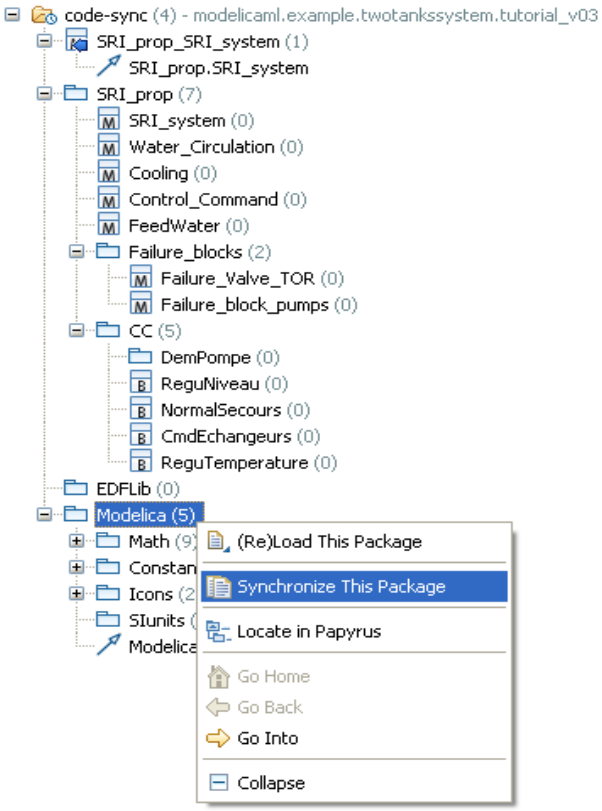

Figure 14 : Illustration of the importation of Modelica models

Now all the models are available in the ModelicaML tool, it is possible to interconnect these models. This is called the "Value Bindings" process and appears as the main specificity brought by ModelicaML tool.

- Test scenarios are linked to Properties. When a test scenario is launched on the process and control models, the properties that are connected to the test scenario are evaluated.
- Properties model are connected to the variables of the states supervisors and/or the models of the physical processes and control systems.
- Test scenarios models are connected to the variables of the states supervisors and/or the models of the physical processes control systems.

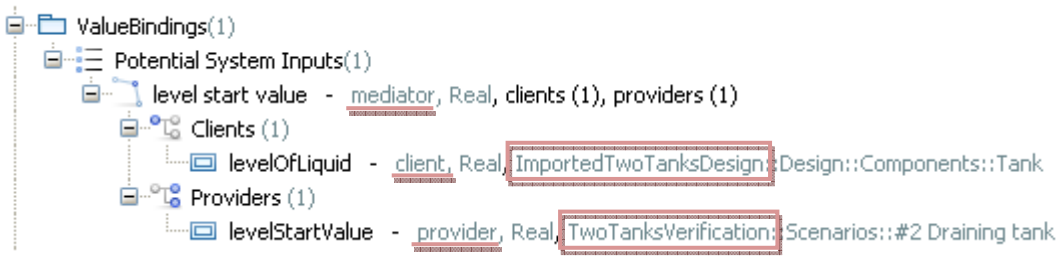

Figure 15 : Example of a value binding process

In the figure above, a mediator (which is a kind of ModelicaML connector) is set between the variable *levelOfLiquid* and the variable *levelStartValue*. *levelOfLiquid* comes from the imported process model whereas *levelStartValue* has been created during the definition of the test scenario.

We can notice that both variables have different labels even though they have to be connected. This facilitates independency between design and validation activities.

As a result we get the following models architecture in ModelicaML tool:

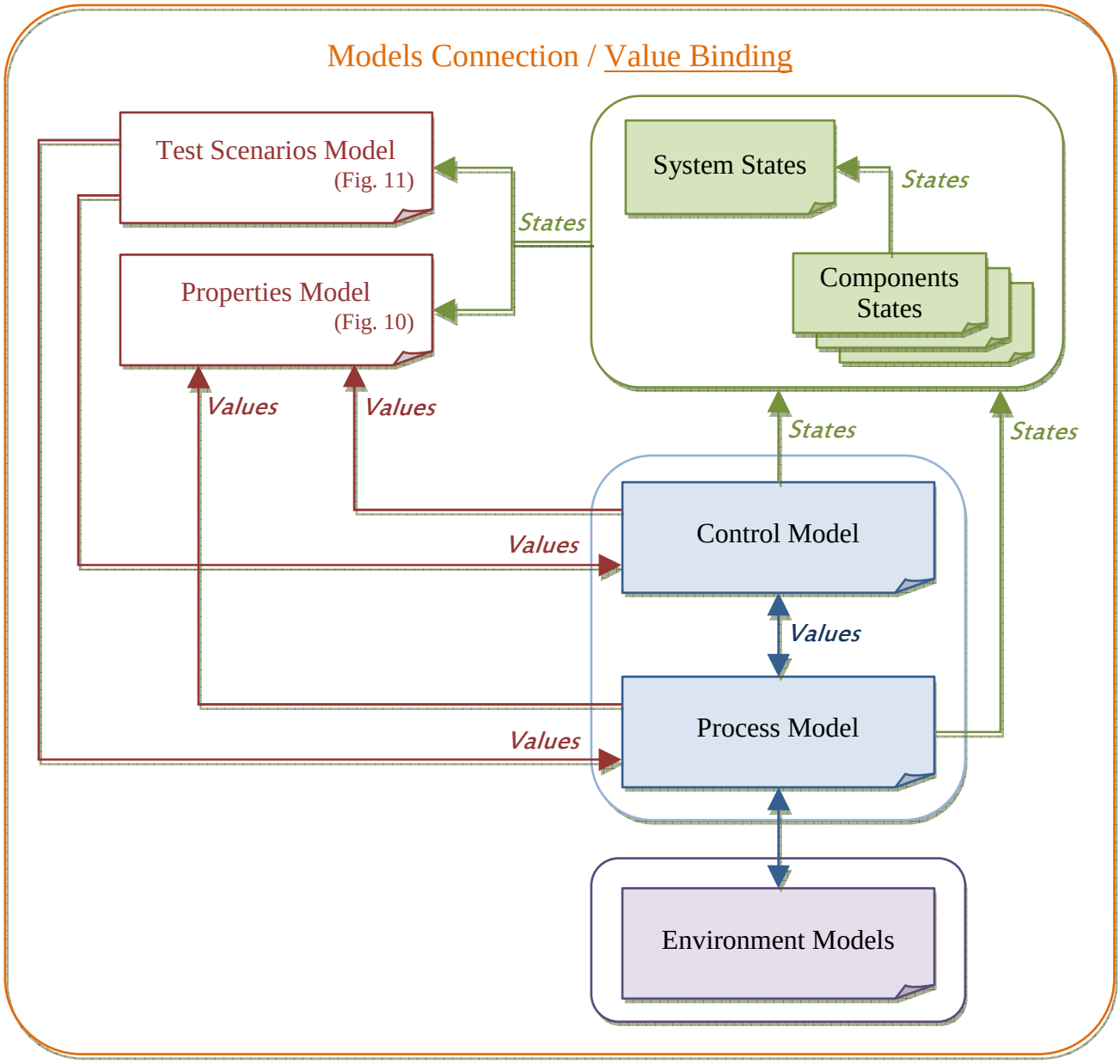

Figure 16 : Values and States Binding / Connections

When all models have been connected, this is possible to launch the simulation. At this time, the properties model, the test scenarios model and the state supervisor model are converted into Modelica files (*.mo). The OpenModelica simulation environment is called. Models and simulation parameters are transferred to OpenModelica that performs the simulation and sends back data results to the ModelicaML Eclipse environment.

Finally, the ModelicaML environment provides plotting and report capabilities so as to process simulation results. Thanks to these features, this is possible to determine if properties were evaluated, violated, over which period etc.

As a digest of the steps presented above, here is presented a scheme of the complete ModelicaML methodology. Alternative methodologies are possible.

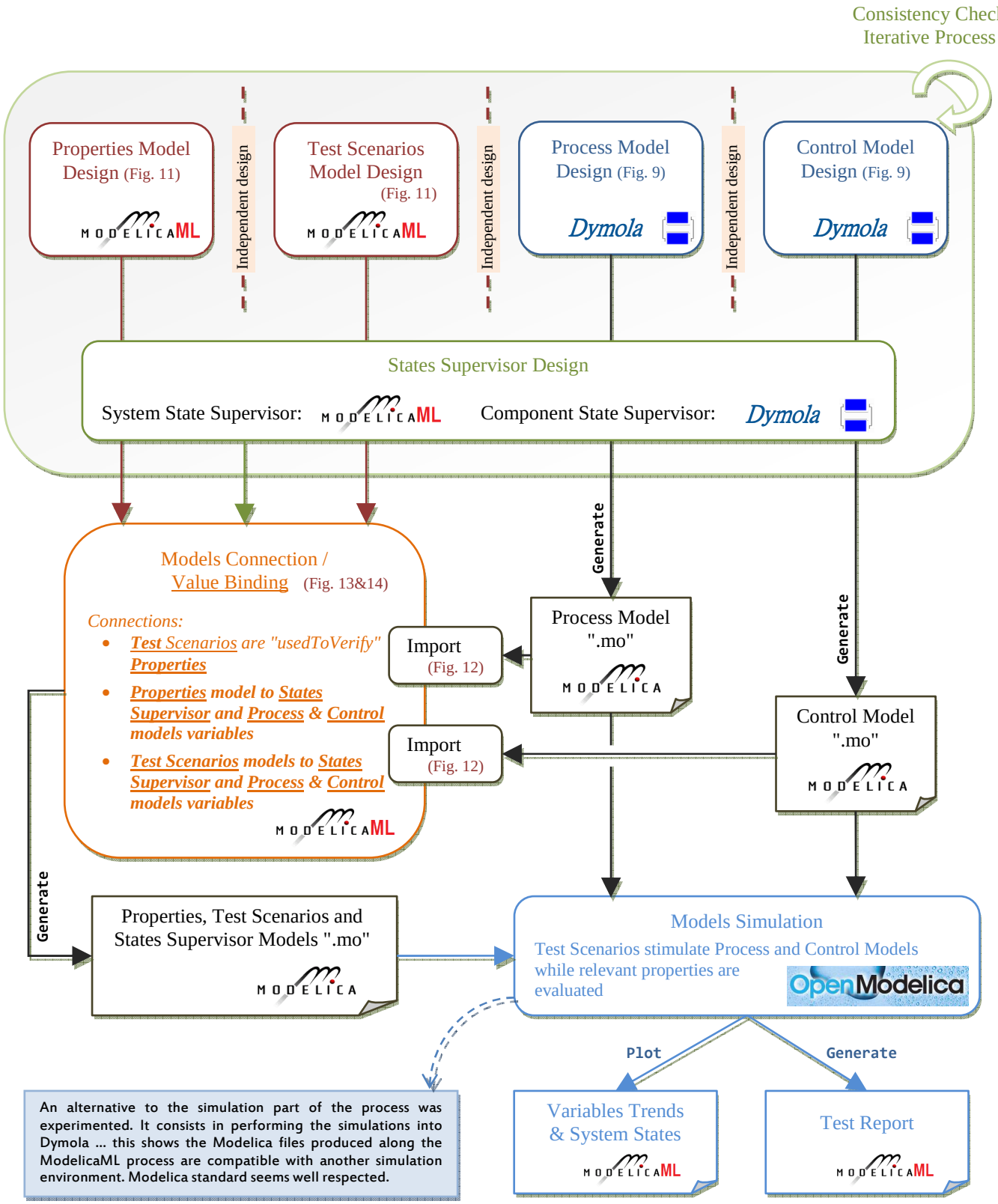

Figure 17 : ModelicaML methodology

In the figure above, tools are presented for each activity. This is a proposition which only aims at illustrating a scenario. Many solutions could be theoretically implemented.

6. Conclusion

6.1. Work performed within OPENPROD

We managed to follow roughly the tutorial document thus recreating small examples in the ModelicaML tool. This was not so easy due to the software which is not really user-friendly for engineers in physics. We managed to go a bit further importing and exporting models (from Dymola to OpenModelica) via the ModelicaML environment. We created new properties, imported external libraries and performed few simulations with imported models.

However we could not implement the SRI study case in ModelicaML. The reasons are dealt in the following evaluation chapter.

6.2. Evaluation of ModelicaML

6.2.1. Pros

ModelicaML keeps and improves the capabilities of Modelica regarding properties and test scenarios modeling. Indeed, ModelicaML provides textual programming based on the Modelica language and adds some additional features like statechart diagrams which are implemented to ease the evaluation of properties and the definition of tests scenarios.

Moreover, test scenarios and properties are clearly linked with the formalized relationship: "usedToVerify". Verification Scenarios are used to verify Properties. This is then possible to setup a cross relationship between Test Scenarios and Properties (which scenario is involved in which property evaluation). This relationship is easily configurable (and tunable) through diagrams that look like SysML Requirements diagrams.

ModelicaML tool comes up with a methodology, which is detailed above in the document on Figure 17. This methodology is based on the independent and concurrent design of system models and test models. The methodology is both based on simple principles and easy to follow.

As an example, value binding is more efficient in ModelicaML Tool (intuitive call of variables for connection) than in Dymola (connections are performed through a manual drawing which can be more complicated to do and to maintain).

ModelicaML tool also produces automatically elaborated test reports. This feature, which is missing in Dymola, has not been evaluated deeply yet.

ModelicaML tool framework is extremely open to the Modelica world which makes it possible to imagine different use cases for the tool. Here are given some examples:

- It could be possible to use ModelicaML tool to perform the modeling of every single part of the models. Indeed, the EDF libraries could be imported and all the design could be performed inside the ModelicaML tool. However, it would probably be extremely laborious because the tool is not user-friendly at all which makes it mandatory for the user to get familiar with the Eclipse programming framework. Besides, the ModelicaML tool does not embed model initialization capabilities.
- Since the tool is complicated to use, it seems smarter to reserve it for tasks on which it has some added value (compared with Dymola). Then, it would be more realistic to use ModelicaML tool only to develop properties, program test scenarios and perform value binding between properties, test scenarios and simulation models. Doing this would clearly separate validation activities from design activities.
- One could also imagine a very light use of the ModelicaML tool for value binding only. The ModelicaML tool could import every compatible model (simulation, tests, and properties) so as to setup (manually) and generate (automatically) the relevant value binding ".mo" file.
- The ModelicaML tool calls OpenModelica for simulation work but theoretically, it is possible to take the

generated ".mo" files (properties, scenarios, value binding) and to load them manually into Dymola or any other Modelica compatible tool for simulation. Nevertheless, each case may need a dedicated evaluation in order to guarantee the feasibility.

Finally, the ModelicaML tool being based on the Eclipse framework, this also paves the way to the implementation of many possible evolutions. Indeed, many open source plug-in's are available in the Eclipse community. One could imagine implementing requirement traceability, configuration management, database storing, report generation and many other features.

6.2.2. Mitigations and Cons

The ModelicaML language does not permit to express new kinds of properties. Indeed, ModelicaML makes a direct use of Modelica language and provides an activity diagram that does not bring additional features. It provides an alternative way to describe only a subpart of what is possible directly with the Modelica language.

Some features are still missing like space locators.

Moreover, the tool, which is promising, is still quite far from being user-friendly. Indeed:

- Menus are overloaded with lower development layers of Eclipse,
- The stability on computers that were chosen for evaluation was pretty random,
- Installation was not possible on every computer without knowing why.

These different problems kept us from going as far as expected in the tool evaluation. ModelicaML is oriented to software engineering and may not be adapted to system engineers and process engineers.

6.3. Perspectives

The evaluation of the ModelicaML environment has been a good opportunity to understand and test a methodology that seems pretty interesting for model-based V&V activities. This permitted EDF to go deeper into the identification of its needs regarding the modeling of property and tests scenarios. Concepts related to the test process (Property, State Supervisor, Behavior model, Environment model, Scenario …) and their relationships are being defined and shared at EDF. This will participate to better structure the test environment that is studied for enforcing model-based validation activities.

The work will carry on in the framework of the INCOME and CONNEXION projects and will also be part of discussions around the next versions of the Modelica language through the upcoming ITEA2 MODRIO project.

The integration of the test process introduced here in the full I&C development lifecycle shall also be addressed. This could be started by linking Properties as described here to high level requirements.

On the ModelicaML tool side, despite its lacks that are completely understandable due to the fact that it is not a commercial tool and relatively recent, it is probably interesting to keep us well informed of its evolutions. ModelicaML tool could indeed be considered as an incubating environment for good ideas.