A SysML Formal Framework to Combine Discrete and Continuous Simulation for Testing

Jean-Marie Gauthier, Fabrice Bouquet,
Ahmed Hammad, and Fabien Peureux

Institut FEMTO-ST – UMR CNRS 6174, Univ. Bourgogne Franche-Comté
16, route de Gray, 25030 Besançon, France
{jmgauthi,fbouquet,ahammad,fpeureux}@femto-st.fr

Abstract. The increasing interactions between huge amount of software and hardware subsystem (hydraulics, mechanics, electronics, etc.) lead to a new kind of complexity that is difficult to manage during the validation of safety-critical and complex embedded systems. This paper introduces a formal SysML-based framework to combine both discrete and continuous simulation to validate physical systems at the early stage of development. This original modelling framework takes as input a SysML model annotated with Modelica code and OCL constraints. Such a model provides a precise and unambiguous description of the designed system and its environment, involving both discrete and continuous features. This formal framework enables to automatically generate Modelica code to perform real-time simulation. On the basis of a constraint system derived from the discrete SysML/OCL modelling artefacts, it also makes it possible to automatically generate black-box test cases that can be used to validate the simulated system as well as the corresponding physical device. This framework has been validated by conclusive experiments conducted to prototype a new energy manager system for aeronautics.

Keywords: SysML, Model-Driven Engineering, Real-Time System, Discrete & Continuous Simulation, Modelica, Constraint Solving, Model-Based Testing.

1 Introduction

Due to increasing behavioural complexity and growing technology heterogeneity combined with still higher expectations, checking that a software embedded system meets its specifications becomes more and more complex, expansive and time-consuming. Moreover, in the traditional development of such systems, the Verification and Validation (V&V) activities begin only after implementation and integration are completed. Under these conditions, discovered problems are particularly more difficult and more expensive to fix, what is a major concern especially when the systems are critical, such as lots of system for aeronautical, railway, automotive, nuclear or telecommunication domains. In these contexts, such systems indeed require to be as trusty as possible because the most little failure could lead to financial as well as human losses, and even so to an irreversible damage of the whole system including its environment.

To mitigate these issues, Model-Based Software Engineering (MBSE) approaches have emerged for several years as a way to improve and automate design, analysis, development, verification and validation of the software embedded in high technology products. Basically, MBSE aims to achieve these software life cycle activities using models that describe the system under development. This kind of approach is mostly supported using (semi-)formal modelling artefacts, which are enough precise to achieve formal verification, but also simulation and testing that provide early practical feedback to validate requirements [1]. Simulation code generation from formal model is increasing as it reduces the gap between high level of abstraction modelling and rapid prototyping, as demonstrated in [2]. Finally, using formal model also enables to apply Model-Based Testing (MBT) approaches [3] that aim to cross-check a model against an implementation, and hence make it possible to provide early validation of functional as well as non-functional properties, such as performance and resource use.

In addition, applying iterative and incremental approaches has also helped the development of critical embedded systems, especially within real-time domain. Such typical approaches are known as *In-the-Loop* processes, and can be performed at different levels: Model-in-the-loop (MIL), hardware-in-the-loop (HIL), processor-in-the-loop (PIL), and software-in-the-loop (SIL) [4]. Simulation and testing are at the core of all these system design processes. For example, within MIL process, at the early stages of the design process, the system (or subpart of the system) and its environment are modelled and simulated using languages such as Modelica[1] or Matlab-Simulink[2] to ensure that the designed (sub)system conforms to its requirements [5]. Another level of simulation and testing concerns the HIL process and consists to test the real hardware platform in combination with its simulated environment (called the *plant* model) [6].

This paper describes an original SysML-based formal framework for simulation and testing of multi-physical and critical systems, that bridges the gap between high-level design model, starting point of MBSE approaches, and real-time execution platform, keystone of the In-the-Loop approaches. In this way, this framework allows system engineers to stay as close as possible of the initial design specifications when achieving all the steps of the development life-cycle. Moreover, it takes advantage of both approaches by ensuring a model centric process enabling validation, simulation and testing from the earliest stage of design. To achieve that, the architecture and the discrete behaviour of the system are described by a Systems Modeling Language (SysML) [7] model, which is annotated with OCL and Modelica code to specify its discrete and continuous features. This model is used to automatically generate real-time Modelica program for simulation, and black-box test cases for validation. The generated test cases can be simulated using the generated Modelica program to validate the design model as well as the the physical system itself. Therefore, the proposed framework can contribute both to MIL process (model against simulated environment), and to HIL process (physical system against simulated environment).

[1] `https://www.modelica.org/documents/ModelicaSpec33.pdf`
[2] `http://www.mathworks.fr/products/matlab/`

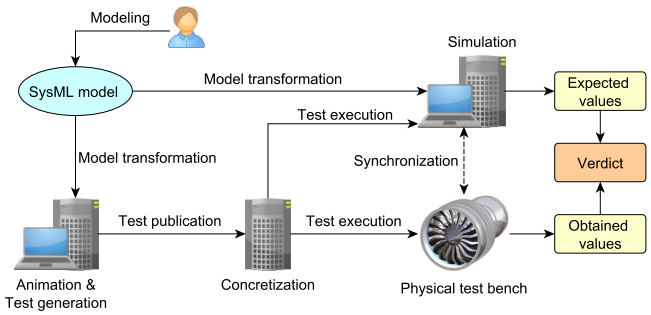

Fig. 1: Overview of the Validation Process from SysML Models

The validation process supporting this formal framework is depicted in Fig. 1. The process starts on the top left with the SysML model that specifies both the system and the plant. This model is expressed using a dedicated subset of the SysML language, which integrates as a whole SysML constructs, Modelica code and OCL annotations. From such a model, a first transformation automatically produces an executable Modelica program to perform real-time simulation. A second transformation allows deriving a set of constraints describing the discrete and abstract behaviours of the system. This set of constraints can be solved to achieve (in a discrete manner) animation of the system as well as test case generation by selecting a subset of trace executions. These abstract test cases are concretized into executable test scripts that can be executed both on Modelica simulation model (within MIL and HIL processes) and physical test bench (within HIL process). During MIL testing, simulation results are manually compared with the initial system requirements and may also be assessed by the domain experts. Once the model is validated, it becomes the test oracle within the HIL testing process: it computes the expected values of the test cases and allows systematic comparison with the real values obtained on the physical test bench (a synchronization step between simulation and test bench environments is required to automate the comparison and the verdict assignment).

The paper is organized as follows. Section 2 introduces the background of the framework to achieve modelling, animation, simulation and test generation from SysML models, and motivates the work presented in the paper. In Sect. 3, we detail the SysML and OCL subset supported by the framework for discrete specifications, and we describe how it is natively combined with Modelica for real-time simulation. Section 4 reports on the conclusive results obtained on a real-life case-study about an helicopter Energy Manager System. Finally, after discussing related work in Sect. 5, we conclude and outline future work in Sect. 6.

2 Background and Motivation

This section clarifies our motivation by introducing preliminaries on the SysML modelling language and the Modelica simulation code, and presents the standard SysML4Modelica that specifies the way to combine them. We also describe the test engine that enables to animate SysML models and apply MBT strategies.

2.1 SysML Modelling Language

The SysML Modelling Language [7], developed within the Object Management Group (OMG) since 2001, enables system engineers to specify all aspects of a complex system using graphical constructs. SysML is a UML profile that adapts the UML semantics to the system engineering field. The semantics of UML, through class and composite structure diagrams, has been moved to the system-level in SysML by the definition of the Block Definition Diagram (BDD) and Internal Block Diagram (IBD). The BDD is based on the UML class diagram. It enables to define component using blocks and their relationships such as associations, generalizations and dependencies. These blocks are instantiated as parts in the IBD, which is a system-level version of the UML composite structure diagram. It specifies the internal organization of a block by describing its parts and the connections between them. Usually, parts are connected through flow ports.

Using SysML allows engineers to achieve MBSE approach to specify, develop and maintain complex systems, notably in the aerospace industry as shown in [8]. In previous work [9], we have proposed to use SysML models to apply MBT strategies to automatically generate functional black-box test cases. To reach this goal, subsets of SysML and OCL, called SysML4MBT and OCL4MBT, have been defined to precisely model the expected behaviour of the System Under Test (SUT) [10]. It contains BDD and IBD to specify the static structure of the SUT and its environment, and state machines with OCL constraints to specify, in a discrete way, behavioural features. Such models are complete and precise enough to automatically derive black-box test cases using the CLPS-BZ test engine.

2.2 Model Animation and Test Generation using CLPS-BZ

CLPS-BZ [11] is a constraint solver that augments the capabilities of (and cooperates with) the integer finite domain solver of SICStus Prolog[3] by handling constraints over sets, relations and mappings. Initially built to animate and generate test cases from B and Z set-oriented formal specifications [12], it has been extended to manage object-oriented specifications, such as UML/SysML models with OCL constraints [13]. Basically, such models are translated into an internal Prolog-readable syntax, called BZP, which provides special constructs for defining SysML diagrams and OCL expressions as constraints over sets. CLPS-BZ makes it possible to efficiently execute on discrete domains the BZP code, both for model animation and for test computation. Test computation consists to look into the graph of reachable states of the system described by the constraints to achieve classical test coverage criteria including transition-based, decision-based and data-oriented criteria [9]. Afterwards, a set of execution traces, that define the test cases, is computed by solving the constraints to find the sequences of operation invocations that ensure the given criteria. To achieve that, CLPS-BZ animates the model and computes a reachability graph, whose nodes are the constrained states built during the animation, and whose transitions define an operation invocation. Using constraint solving dramatically reduces the search space during test generation, which allows the method to scale to larger systems.

[3] `https://sicstus.sics.se`

The constraint system, described using the BZP format, obviously defines a Constraint Satisfaction Problem (CSP) [14], i.e. a set of constraints, which must be satisfied by the solution of the problem it models. Formally, a CSP is a triplet $< V, D, C >$ where V is a set of variables $\{v_1, \ldots, v_n\}$, D is a set of domains $\{d_1, \ldots, d_n\}$, where d_i is the domain associated with the variable v_i, and C is a set of constraints $\{c_1(V_1), \ldots, c_m(V_m)\}$, where a constraint c_j involves a subset V_j of the variables of V. Within CLPS-BZ, which is able to manage sets and integer finite domains, variables of V can be either an *atom*, or a set of atoms ($set(atom)$), or a set of (nested) pairs of atom ($set(pair(atom, atom))$). However, the CLPS-BZ technology is only able to handle constraints on discrete domains and thus can execute neither animation nor test generation based on continuous formula to efficiently address real-time systems. Therefore, CLPS-BZ enables to derive test cases, as sequences of operation invocations, but an other and independent model or program is necessary to execute them in the continuous domain to gather the real and expected results. Moreover, SysML is natively not executable: it does not include an action language, which could allow to simulate SysML model, and even less if equations occur. To overcome this lack, the OMG has proposed an extension to SysML to allow clarifying such mathematical properties into SysML models using Modelica code. Hence, we propose to use this extension to adapt and complete the existing approach to be able to manage in a single model both high-level discrete requirements and low-level continuous behaviours for test generation purpose.

2.3 Modelica and SysML4Modelica

Modelica is an object-oriented and equation-based language adapted to complex physical systems modelling. Indeed, Modelica is built on acausal modelling with mathematical equations and object-oriented constructs, and is designed to support effective library development and model exchange. Since 2012, the OMG promotes a dedicated SysML-Modelica Transformation specification[4] to integrate Modelica semantics into SysML and to provide a bi-directional transformation between the both languages. The specification gives an extension to SysML, called SysML4Modelica, which proposes matching semantics between the SysML constructs and the Modelica code. The integration of Modelica concepts into SysML is based on profiling: the SysML4Modelica constructs enable to stereotype elements, which are parts of the BDD and the IBD of SysML.

Hence, to describe complex and heterogeneous systems, the SysML4Modelica profile enables to bring together, in a single model, the non executable graphical high-level SysML modelling and the real-time and continuous Modelica specifications. However, no theoretical framework is given to provide a practical way to combine the architecture and discrete behaviours of SysML models with the continuous aspects described by Modelica formula. This paper bridges this gap by defining such a framework to bring them back together to achieve model-based testing. It integrates constraint solving to address discrete animation and black-box test generation, and Modelica simulation to address continuous needs.

[4] `http://www.omg.org/spec/SyM/`

This proposed framework aims (1) to avoid managing several models (at least one for high-level discrete design and one for low-level continuous features) that require to be manually synchronized, (2) to increase the automation level of the model-based testing approach by minimizing the number of testing artefacts and by providing a native link between abstract data (from SysML structures) and executable structures (derived from Modelica code), and (3) to foster the use of MBSE approach by supporting in the same modelling framework all design steps of the real-time system life-cycle activities. The next section precisely introduces the modelling framework we define to efficiently combine discrete and continuous features in a single model for simulation and model-based testing purposes.

3 Combining Continuous and Discrete Modelling

This section gives a formal description of the SysML modelling framework unifying discrete and continuous points of view. As depicted in Fig. 2, the resulting framework is defined by the intersection of the SysML subset for discrete modelling with the one dedicated to continuous modelling. In this section, these both SysML subsets are detailed and we show how the combination of them defines a formal SysML modelling framework for simulation and testing activities. Afterwards, the next section will introduce a proof-of-concept integrated tool chain implementing this theoretical framework. However, this framework defines a generic way to combine discrete and continuous SysML modelling, and therefore it can be used and implemented using another similar tooled approaches.

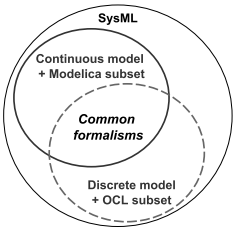

Fig. 2: Overlap of the SysML subsets

NOTE: in the rest of the paper, to dispel any ambiguity and avoid misunderstanding, *animation* is defined as a discrete evaluation of the model (variables belong to finite domains or sets), i.e. an execution of the model based on constraint solving restricted to the SysML data that belong to finite domains or sets, whereas *simulation* means real-time simulation of the model, i.e. an execution of the Modelica code describing the system in a continuous-time process.

3.1 SysML Subset for Simulation

The SysML subset for simulation purpose focuses on the following diagrams: BDD, IBD and state machine. The structural view of the system is specified in the BDD with blocks, which are connected each other using flow ports that are depicted in the IBD. The behaviour of each system component may be described using state machines. We thus define a SysML model for simulation as a model M_s comprising two kind of blocks (blocks for component definition and blocks that type flow ports) and enumerations. A block that types flow ports only contains properties (no behaviour). SysML enumerations enable declaring abstract types that can be used during a Modelica simulation.

Definition 1 (Model for simulation). *Let M_s, the model for simulation, be given by $M_s =< \eta, \Gamma_{Bs}, \Gamma_{Bf}, \Gamma_{Enum} >$, where η is the name of the model, Γ_{Bs} is the set of SysML blocks that defines components, Γ_{Bf} is the set of SysML blocks that types flow ports and Γ_{Enum} is the set of enumerations.*

A component is defined with attributes and may be composed of other components. Thus, it may have different typed elements (properties, parts and flow ports). Its behaviour may be specified by a state diagram with Modelica code.

Definition 2 (Block for component definition). *We define $\beta \in \Gamma_{Bs}$ to be the tuple $\beta =< \eta, \Gamma_{Att}, \Gamma_{Part}, \Gamma_{FP}, \Gamma_{Cnt}, \Gamma_{Cons}, \Gamma_{SM} >$, where:*

1. *η is the unique name of the block,*
2. *Γ_{Att} is the set of attributes,*
3. *Γ_{Part} is the set of parts,*
4. *Γ_{FP} is the set of flow ports,*
5. *Γ_{Cnt} is the set of connectors,*
6. *Γ_{Cons} is the set of constraints,*
7. *Γ_{SM} is the set of parallel state machines.*

Each attribute, each part and each flow port shall be typed with primitive types (real, integer and Boolean, respectively noted $\mathbb{R}$, $\mathbb{Z}$ and $\mathbb{B}$) or with user-defined type (block for part, block for flow port and enumeration, respectively noted Γ_{Bs}, Γ_{Bf} and Γ_{Enum}). The set of types Γ_T by $\Gamma_T = \{\Gamma_{Bs}, \Gamma_{Bf}, \Gamma_{Enum}, \mathbb{R}, \mathbb{B}, \mathbb{Z}\}$ are defined as follows. Concerning attributes of Γ_{Att}, we have to distinguish several cases: an attribute may be a constant, an equation's unknown or a parameter that defines the initial condition of the simulation. Within Modelica, an equation's unknown, which needs to be solved by integration, can be either continuous or discrete.

Definition 3 (Attribute). *Let $\alpha \in \Gamma_{Att}$ be defined by $\alpha =< \eta, \omega, \upsilon, t >$ where:*

1. *η is the name of the attribute,*
2. *ω is variability such as $\omega \in \{$constant, parameter, discrete, continuous$\}$,*
3. *υ is the value of the attribute,*
4. *t is the type of the attribute (Γ_{Enum}, $\mathbb{R}$, $\mathbb{B}$, or $\mathbb{Z}$).*

If an attribute is discrete or continuous, then it is necessarily a state variable.

A block that types ports can only have properties, i.e., attributes that describe what flows between ports. Then, the following is the definition of such blocks:

Definition 4 (Block for flow ports typing). *We define $\beta_f \in \Gamma_{Bf}$ to be the tuple: $\beta_f =< \eta, \Gamma_{Att} >$, where η is the unique name of the block and Γ_{Att} is the non-empty set of attributes.*

We need also to formalize the connection between parts of a SysML model. Connections are always between two flow ports, and a flow port has to be connected at least to one other flow port. Then, we define the surjective connecting function as follows:

Definition 5 (Connecting function). *Let f_c, the surjective connecting function, be defined by $f_c : \Gamma_{FP} \times \Gamma_{FP} \twoheadrightarrow \Gamma_{Cnt}$.*

The continuous behaviour of the system is specified by equations over continuous state variables. The SysML constraints (Γ_{Cons}) are written using a subset of the Modelica language that expresses equations. Numerical solvers (embedded in all Modelica frameworks) are able to rewrite such constraints into a set of first-order differential equations in order to compute integration over time.

Finally, state machine diagrams enable to describe the life-cycle of a component. For instance, one may specify several component states depending on time, state variables or user behaviours. The formal definition is given below (it excludes join, fork and history pseudo-states that are not supported).

Definition 6 (State machine). *State machine for simulation is described with its classical definition* $SM =< s_0, \Sigma, \Gamma_E, \mathscr{L}_s, \delta >$*, where:*

1. s_0 *is the initial state,*
2. Σ *is a finite non-empty set of states composed of three disjoint sets: simple states* Σ_{ss}*, compound states* Σ_{cs} *and eventually final states* Σ_{fs}*,*
3. Γ_E *is the set of trigger events,*
4. $\mathscr{L}_s$ *is the alphabet for specifying guard and effect of a transition,*
5. $\delta : \Sigma \times \Gamma_E \times \mathscr{L}_s \rightarrow \Sigma$ *is the transition function.*

The language $\mathscr{L}_s$, used for specifying guards and effects, is a subset of the Modelica language. A guard is a Modelica Boolean expression and an effect is a Modelica statement such as *assignment*, *if-statement*, *while-statement* or *for-statement*. Moreover, each state may have *onEntry* and *onExit* actions, which are respectively executed at the entry and the exit of the state. These are defined using Modelica statements. Concerning trigger events Γ_E, we only consider call events, i.e representing an operation call. The called operations have to be defined in the block that the state machine specifies. However, we do not consider the operations of the blocks for components definition because operations are not translated into Modelica code, only trigger events are.

The above presented subset (summarized in Table 1 in Sect. 3.3) is sufficient to perform Modelica code generation and simulation. This subset enables to validate a system at the earliest stage of a design process by automating the derivation of Modelica code. To provide a modelling framework that enables to perform both simulation and testing from a single SysML model, discrete aspects of the system have also to be integrated in order to use the CLPS-BZ solver for animation and test case generation purposes.

3.2 SysML Subset for Animation and Test Generation

The SysML subset for animation (SysML4MBT with OCL4MBT [9]) enables to formally specify the system to perform a constraint evaluation. Such a SysML model describes the system from an abstract and discrete point of view. The model is abstract in the way that the domain of a variable in $\mathbb{R}$ is discretized using enumeration classes since CLPS-BZ only manages integers, Booleans and finite sets. The behaviour of the model is also discrete as, during animation, we do not know what happen between two stable states of the state machines. Of course, simulation gives us some information about it, but during model animation, each state transition is executed as an atomic and non-breaking computation.

Definition 7 (Model for animation). *The model for animation M_a is defined by $M_a =< \eta, \Gamma_{Ba}, \Gamma_{Enum}, \Gamma_{Asso} >$, where η is the name of the model, Γ_{Ba} is the set of SysML blocks that defines components, Γ_{Enum} is the set of enumerations and Γ_{Asso} is the set of associations between blocks.*

Associations of Γ_{Asso} are translated into relations between instances of classes. The multiplicities of the association determine whether the relationship is a function, and if so, the type of this function (partial or total, and possibly injective, surjective or bijective). Hence, the associations are translated into structures of type $\Gamma_{Part} \times \Gamma_{Part}$.

A block for component definition comprises attributes, parts and operations, which are used to describe actions from the environment.

Definition 8 (Block for component definition). *We define $\beta \in \Gamma_{Ba}$ to be the tuple $\beta =< \eta, \Gamma_{Att}, \Gamma_{Part}, \Gamma_{Op}, \Gamma_{SM} >$, where:*
1. *η is the unique name of the block ,*
2. *Γ_{Att} is the set of attributes,*
3. *Γ_{Part} is the set of parts,*
4. *Γ_{Op} is the set of operations,*
5. *Γ_{SM} is the set of parallel state machines.*

Blocks define variables of the CSP and their domains are defined by the set of instances (Γ_{Part}) of these blocks. With CLPS-BZ, each block is associated with information concerning its instances: the set of instances that can potentially be created (*all_instances* as a $set(atom)$), the set of currently created instances (*instances* as a $set(atom)$), and the current instance, which is the last created instance or treated by an operation (*currentInstance* as an $atom$). Among all the possible instances *all_instances*, a fictitious *none* instance is created. It is used in the case to formalize the absence of current instance. Concerning enumerations, they are translated into $set(atom)$, where the atoms are the literals defined in the enumeration. Thus, enumerations define domains in the CSP.

For animation, the set of types Γ_{Ta} is defined by $\Gamma_{Ta} = \{\Gamma_{Ba}, \Gamma_{Enum}, \mathbb{B}, \mathbb{Z}\}$.

Definition 9 (Attribute). *Let $\alpha \in \Gamma_{Att}$ be defined by $\alpha =< \eta, \omega, \upsilon, t >$ where:*
1. *η is the name of the attribute,*
2. *$\omega \in \{$constant, variable$\}$, and if $\omega =$ variable then ω is a state variable,*
3. *υ is the value of the attribute during the animation,*
4. *t is the type of the attribute ($\mathbb{Z}$, $\mathbb{B}$ or Γ_{Enum}).*

Each attribute $\alpha \in \Gamma_{Att}$, belonging to a block β, is translated into a total function between all instances of the block β and the domain of α. Considering for example that α is an integer, α is translated into a structure of type $\Gamma_{Part} \times \mathbb{Z}$.

The operations have a name and optional parameters (in, out, $inout$, $return$). For animation purpose, we only take into account in and $return$ parameters. In addition, operations can also have OCL4MBT precondition and postcondition.

Definition 10 (Operation). *Let $o \in \Gamma_{Op}$ defined as $o =< \eta, \Gamma_{Par}, pre, post >$ where: η is the name of the attribute, Γ_{Par} is the set of parameters, pre is the precondition of the operation and post is the postcondition of the operation.*

State machines are used to specify discrete component behaviours and external, physical or human, actions. For animation purpose, state machines are defined as expressed in the definition 6. However, we define $\mathscr{L}_a$, based on the OCL4MBT subset, as the alphabet for specifying guard and effect of a transition.

Each state (single, composite, initial or final state) of a state machine is translated into a specific context of the CSP. For each state, a variable *status* stores the current state(s) of a block instance: it is a function associating each instance of the block to a Boolean (the function is partial due to the presence of the fictitious *none* instance in its domain). At the beginning of the animation, each instance is in the initial state. In addition, two operations are declared to each state to formalize the possible *onEntry* and *onExit* effects.

Each state machine is associated with the block it specifies the behaviour. Operations of this block can be used as triggers for some transitions of the state machine. To avoid unmanageable infinite loop during animation, three types of transitions are allowed: external (reflexive or not) with trigger, internal with trigger and guarded external or automatic (not reflexive). A variable $opCalled \in \Gamma_E$, declared to store the last executed operation, enables to fire, from the current state, transition triggered by this operation. We finally add a precondition for all guarded and automatic transitions, expressing that no operation has been called ($opCalled = none$). This ensures that the UML "run-to-completion" semantic is satisfied. To sum up, each state gives rise to a variable *status* and constraints related to the *onEntry* and *onExit* actions. Each transition is translated into constraints in the CSP that are defined by its guard and effect. Finally, trigger events of Γ_E define a set of operation triggers $set(atom)$ that defines the domain of the variable *opCalled*.

The above formalized subset (summarized in Table 1 in Sect. 3.3) enables to animate a discrete and abstract SysML model by translating it into a CSP. This CSP is defined by a Prolog-readable BZP file such that it is now possible to specify the continuous and discrete behaviour of a complex and critical system for simulation, animation and testing purpose. In the next subsection, we combine these subsets to adress both continuous and discrete features.

3.3 Combined Formalism for Simulation and Animation

Table 1 summarizes the SysML subsets for simulation and animation. Each combined SysML element are derived both to Modelica element and to CSP element (variable V, domain D or constraint C). To propose a unified modelling framework, blocks and enumerations for simulation have to be used for animation. Then, the model for validation M_v is defined as $M_v = M_s \cap M_a = \{\Gamma_{Bv}, \Gamma_{Enum}\}$ where $\Gamma_{Bv} = \Gamma_{Bs} \cap \Gamma_{Ba}$. Blocks for flow port typing (Γ_{Bf}) are not used for animation. Considering now $\beta_1 \in \Gamma_{Bs}$ and $\beta_2 \in \Gamma_{Ba}$, then $\beta_1 \cap \beta_2 =< \eta, \Gamma_{Att}, \Gamma_{Part}, \Gamma_{SM} >$ where each attribute of Γ_{Att} is defined as proposed in definition 9. Indeed, Modelica is able to process discrete and continuous variables whereas the CLPS-BZ solver is not able to manage continuous state variables. Concerning SysML parts, they enable to instantiate Modelica components and to declare block instances in the constraint system.

Table 1: SysML for Modelica Simulation and CSP Animation

SysML elements	**Modelica elements**	**CSP <V,D,C>**
Model M_v	Root Modelica model	CSP model
Blocks Γ_{Bv}	Model	$\Gamma_{Bv} \in V$
Blocks Γ_{Bf}	Connector	-
Enumerations Γ_{Enum}	Enumeration type	$\Gamma_{Enum} \in D$
Attributes Γ_{Att}	Value property	$\Gamma_{Att} \in V$
Constraints Γ_{Cons}	Equation	-
Parts Γ_{Part}	Component	$\Gamma_{Part} \in D$
FlowPorts Γ_{FP}	Port	-
Connectors	Connect equation	-
Op. Precondition $pre \in \Gamma_{Op}$	-	$pre \in C$ ($\mathscr{L}_a$)
Op. Postcondition $post \in \Gamma_{Op}$	-	$post \in C$ ($\mathscr{L}_a$)
Op. parameters Γ_{Param}	-	$\Gamma_{Param} \in D$
State-Machines SM	Algorithm section	-
States Σ	Boolean variable	$status$ variable $\in V$
State $Entry$	Statement ($\mathscr{L}_s$)	$Entry \in C$ ($\mathscr{L}_a$)
State $Exit$	Statement ($\mathscr{L}_s$)	$Exit \in C$ ($\mathscr{L}_a$)
Event triggers Γ_E	Boolean variable	$\Gamma_E \in D$
Transition δ	When statement	-
Transition $guard$	Boolean expr ($\mathscr{L}_s$)	$guard \in C$ ($\mathscr{L}_a$)
Transition $effect$	Statement ($\mathscr{L}_s$)	$effect \in C$ ($\mathscr{L}_a$)

Finally, state machines for simulation and animation are not totally combined. The language for specifying guard and effect of transitions, as well as *onEntry* and *onExit* actions of states, is indeed not fully equivalent. In one case, the language $\mathscr{L}_s$ is a subset of Modelica and in the other case, the language $\mathscr{L}_a$ is a subset of OCL. However, states, transitions and events are used for both simulation and animation. Thus, every state machines are translated both into Modelica code (using formula of $\mathscr{L}_s$) and CSP (using OCL code of $\mathscr{L}_a$). It should be noted that state machines without OCL code and event trigger are not translated into CSP because it would not impact the CSP solving and could even give a under-constrained CSP (and make it non deterministic).

4 Implementation and Case-Study Evaluation

This section discusses the proposed modelling framework regarding an industrial case-study about a large and complex Energy Manager System (EMS) that delivers energy to a new type of helicopter. Figure 3 shows the architecture of our simulation, animation, and testing environment from SysML models. This Eclipse-based tool chain, that instantiates the intended process given in Fig. 1, strongly relies on Model-Driven Architecture (MDA) approach since model transformation and code generation procedures enable to automatically derive the simulation and testing artefacts from the SysML models [15]. Therefore, the SysML model is translated into Modelica simulation code and into a pivot meta-model, named UML4TST, and next into BZP file. Finally, Papyrus[5] is used to support the SysML modelling, OpenModelica[6] computes the simulations, and CLPS-BZ (included as a plugin in our Eclipse environment) generates the test cases that are exported as UTP sequence diagrams in the SysML model.

[5] `https://www.eclipse.org/papyrus/`
[6] `https://www.openmodelica.org`

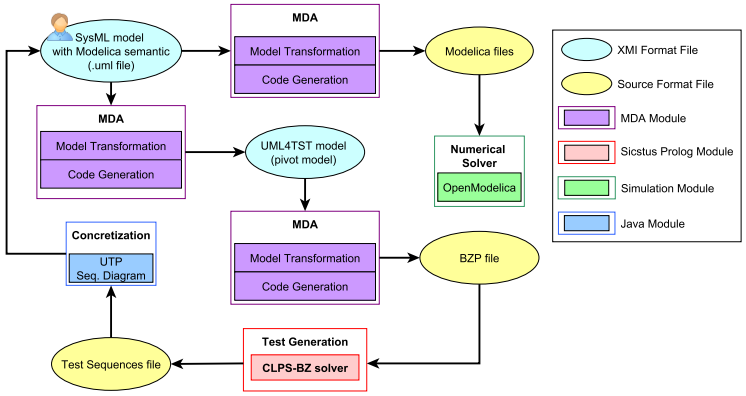

Fig. 3: Overall Architecture of the Simulation and Testing Tool Chain

This tool chain has been tested out during the prototyping phase of the EMS system within HIL process. Hence, we distinguish the system under design (the EMS) and its environment called the plant system (a simulation model of some helicopter's instruments). The objective of our experiment was to assess the suitability and the reliability of the combined formalism to perform simulation and test generation. The experiment started from requirement specifications given in natural language. They describe the EMS and the physical limit of its components, and the instruments of the helicopter with their energy request over time during the activation period. We now describe the main results obtained from this experimentation, which was divided into five stages: (1) EMS and plant modelling (using SysML subset for simulation), (2) simulating the system components using a scenario example, (3) adding abstraction and discrete behaviours (using SysML subset for animation), (4) animating the model and generating test cases, and (5) executing test cases on the simulation model.

SysML Modelling for Simulation. The EMS, depicted in Fig. 4, is composed of an energy source that emulates a permanent power source, an accumulators battery, a battery of super-capacitors, and a bus that connects the energy sources. Each source is described by a specific IBD, which specifies the source and its controller for managing strategies. The helicopter energy requests go through the flow port *Icharge* modelled at the top of the IBD.

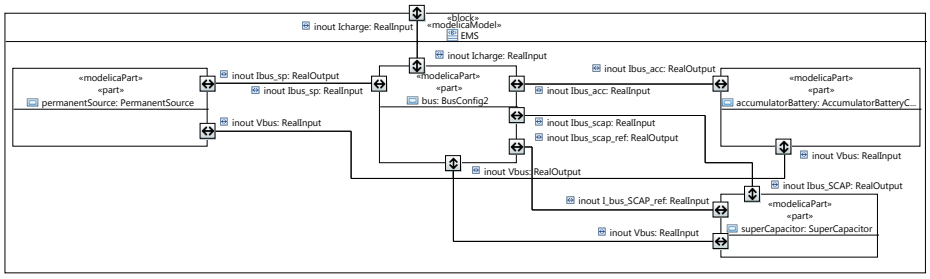

Fig. 4: IBD of the Energy Manager System

The plant model comprises 14 helicopter's instruments that all require energy during a mission. For confidentiality reason, we cannot cite these components and provide more details about the EMS and the plant. However, the IBD of Fig. 5 shows 4 instruments that require energy over time. Each of them is connected to a bus that sums the energy demand. The *outPlant* flow port enables to connect the plant to the *Icharge* flow port of the EMS. The functional and continuous modes of the instruments are specified with state machines and equations.

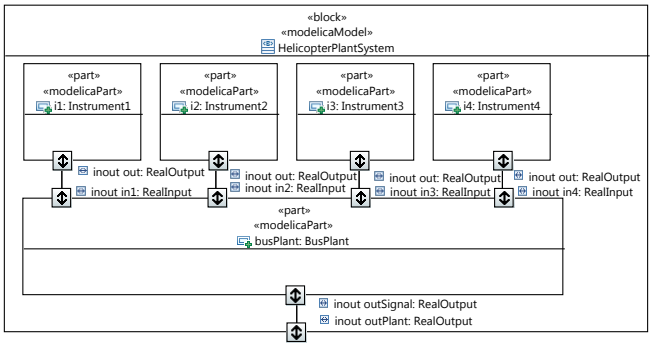

Fig. 5: IBD of the Plant Model

Overall information about the SysML model is provided in Tab. 2. Note that the bold numbers are the same for the EMS and the plant. This means that the EMS and the plant are specified in the same SysML model ($M_s = 1$) and that blocks for flow port typing ($\Gamma_{Bf} = \{RealInput, RealOutput\}$) are used both in the EMS and in the plant. From this SysML model, 626 lines of Modelica code have been automatically generated for the plant, and 412 lines for the EMS.

Table 2: Metrics about the SysML Model

SysML elements	**# for the EMS**	**# for the plant**	**Total**
Model $M_s = M_a$	**1**	**1**	**1**
Blocks $\Gamma_{Bs} = \Gamma_{Ba}$	19	16	35
Blocks Γ_{Bf}	**2**	**2**	**2**
Enumerations Γ_{Enum}	0	1	1
Attributes Γ_{Att}	41	48	89
Constraints Γ_{Cons}	39	15	54
Parts Γ_{Part}	34	15	49
FlowPorts Γ_{FP}	66	30	96
Connectors	76	15	91
State-Machines SM	3	2	5
States Σ	10	21	31
State $Entry$	0	0	0
State $Exit$	0	1	1
Event triggers Γ_E	0	15	15
Transition δ	11	47	58
Transition $guard$	8	16	24
Transition $effect$	0	47	47

Model Animation and Test Generation. The EMS and the plant were translated into a CSP using the BZP format. In this model, no operation preconditions and postconditions were used. The discrete behaviour of the helicopter has been specified only using state machines with OCL4MBT and event triggers.

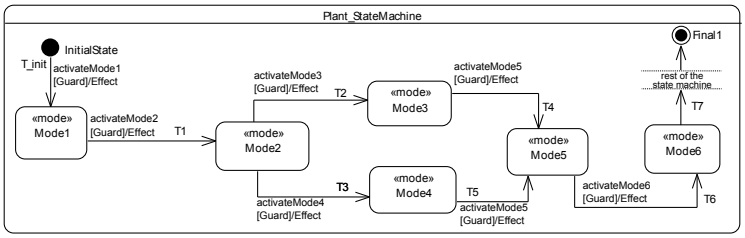

Fig. 6: Excerpt of Plant System State Machine

Basically, a mission of the helicopter is composed of a sequence of several modes. Each mode is an activation of one or more instruments over time that are done by the pilot during the flight. All the possible mode activations have been specified with a state machine, as shown in Fig. 6. The complete state machine contains 17 modes and 55 transitions. Among these 55 transitions, everyone has an event trigger, 17 have OCL4MBT guard and 18 have OCL4MBT effect. OCL4MBT enables to guide the CLPS-BZ solver during test case generation to produce sequences verifying the system requirements. For instance, OCL4MBT code has been added on several transitions to satisfy the following requirement: the mode $Mode9$ (not depicted in Fig. 6) may be activated twice only if the mode $Mode6$ has been activated just before. In addition, each transition effect is completed with Modelica code in order to simulate the instruments during the continuous simulation. From this model, the CLPS-BZ solver has generated 154 test cases to cover all the transitions of the state machine. Each test case is a sequence of 10 to 20 operation invocations. Finally, the concretization step consisted in automatically publishing test cases as sequence diagrams, next translated into Modelica procedures to be simulated.

Feedback and Lessons Learned. First of all, it should be noted that these experiments have been conducted by an engineer with a huge expertise in SysML modelling and model-based testing approach, but without any initial knowledge about real-time simulation and EMS specifications. In this way, it might bias us to have an objective view of the scalability of the process, but it does not affect the suitability of the formal framework we aim to evaluate. Moreover, case-study results have been evaluated with scientists specialized in smart energy systems, who are therefore familiar with development and continuous simulation of such complex systems. This enables us to get a solid feedback regarding the relevance of the framework and the related tool-supported overall approach.

Thanks to these experiments, we can conclude that the proposed modelling framework, combining both discrete and continuous features of the designed system, is relevant to achieve efficient model-based testing. On the one hand, the selected SysML formalism is expressive and precise enough to describe the system, generate relevant abstract test cases, and enable early simulation of the system. On the other hand, the framework offers a concrete benefit regarding the model writing and maintenance since the discrete and continuous features are natively mapped and kept consistent within the SysML model, and they can be automatically checked using test case generation and simulation.

As a consequence, the supporting implementation offers a relevant execution platform for a rapid prototyping and an early validation of the real-time designed system. These experiments have also highlighted the high level of automation regarding test case concretization, which is known to be tricky and time-consuming, especially when real-time constraints occur, as observed in previous work [9] where discrete model and continuous program were distinctly and separately managed and synchronized. This benefit stems again from the native link between discrete model elements (basis of the test generation) and the Modelica code (basis of the simulation). Regarding the process, deeper investigations are required to provide a complete report about scalability and efficiency of the overall approach, in particular w.r.t. industrial practices on large-scale systems.

5 Related Work

To the best of our knowledge, there is no reported approach in the literature that supports continuous and discrete SysML modelling for simulation and testing purposes. However, combining discrete and continuous modelling is not a recent topic. The integration of continuous and discrete aspects for modelling and simulation introduced by Zeigler et al. in [16], which defines a coupled Discrete Event and Differential Equation Specified System formalism (DEV&DESS), is close to our proposal. Basically, a such a coupled model is a model, which contains connected components. Components are defined as atomic DEV&DESS. We first tried to map our SysML modelling framework with this coupled DEV&DESS formalism. But this formalism does not support both discrete and continuous states into a coupled model, so a SysML block with parts cannot support a state machine. Some other results have been provided recent years to achieve similar continuous and discrete modeling for simulation. The approach in [17] proposes to combine superdense time, modal models, generalized functions and constructive semantics to get a rigorous approach for modelling discrete physical phenomena that occur on cyber-physical systems. Nevertheless, this approach does not consider test generation for model and physical system validation.

We have to point out that similar black-box testing approaches exist for real-time systems. For instance, Iqbal et al. [18] propose a modelling methodology based on UML and MARTE, in which the UML model is automatically translated into environment simulators implemented in Java. However, this modelling approach does not deal with continuous aspects, and differential and algebraic equations (DAEs) are hidden to the engineers.

About generation of simulation code from UML and SysML models, in [19], the authors propose to derive VHDL specifications from UML classes and state diagrams. Vanderperren et al. [20] propose to translate SysML models into Matlab-Simulink. Other work [21] focuses on generating SystemC from UML models. Each of these approaches enables to simulate a system specified with UML or SysML, but discrete and abstract aspects of such models are not considered for model-based testing. Moreover, our work is original as we propose to combine continuous and discrete modelling in a single model.

6 Conclusion and Future Work

This paper presented a SysML framework that combines continuous features for simulation and discrete aspects for model-based testing. We formally described the SysML subsets for Modelica simulation and CSP solving, and the way to combine them in a single SysML model. This combined approach aims to be used within model-in-the-loop and hardware-in-the-loop processes. In these contexts, the simulation respectively plays two key roles: simulating a component based system and providing test cases and oracles for the model and its concrete product. While preserving the V cycle to address complex and critical system development, we promote a more iterative and incremental approach driven by the early validation and verification activities. Experiments give a conclusive feedback about the suitability and the reliability of this framework, and highlighted its higher automation ability for early design validation of real-time systems.

As future work, we plan to conduct extensive experiments and to extend CLPS-BZ to handle continuous domain in order to investigate new test generation criteria based not only on discrete features, but also on continuous ones. It would be possible to use the CLPQR library. This library considers real valued variables and enables to perform linear equations solving. Furthermore, we have some insights concerning the combined use of CLPS-BZ with a numerical solver. More precisely, the use of interactive simulation, driven by a numerical solver, would enable to explore the continuous state space between two specified discrete states. This combination requires each solver to manipulate the same object, and requires establishing a communication protocol to propagate deductions made by a solver in the other. Such protocol would not only raise issues about concurrency or synchronization: it would obviously require further investigation about more complex algorithms regarding state reachability issues, including meta-heuristics, patterns recognition, fuzzing, etc. These issues open new research topics combining parallel and distributed fields with formal V&V.

References

1. A. Qamar, C. During, and J. Wikander, "Designing mechatronic systems, a model-based perspective, an attempt to achieve SysML-Matlab/Simulink model integration," in *Int. Conf. on Advanced Intelligent Mechatronics (AIM'09)*. Singapore, Republic of Singapore: IEEE CS, Jul. 2009, pp. 1306–1311.
2. A. Sindico, M. Di Natale, and G. Panci, "Integrating SysML with Simulink using open-source model transformations," in 1^{st} *Int. Conf. on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH'11)*. Noordwijkerhout, The Netherlands: SciTePress, Jul. 2011, pp. 45–56.
3. M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., 2007, ISBN 978-0-08-046648-4.
4. B. M. Broekman, *Testing Enbredded Software*. Addison-Wesley Longman Publishing Co., Inc., 2002, ISBN 0321159861.
5. R. Matinnejad, S. Nejati, L. Briand, and T. Bruckmann, "MiL testing of highly configurable continuous controllers: Scalable search using surrogate models," in 29^{th} *ACM/IEEE Int. Conf. on Automated Software Engineering (ASE'14)*. Vasteras, Sweden: ACM, Sep. 2014, pp. 163–174.

6. A. Benigni and A. Monti, "Development of a platform for hardware in the loop testing of network controller," in *2011 Grand Challenges on Modeling and Simulation Conference (GCMS'11)*. Hague, Netherlands: Society for Modeling And Simulation Int., Jun. 2011, pp. 124–128.
7. S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, 2009, ISBN 978-0-12-374379-4.
8. H. Graves and Y. Bijan, "Using formal methods with SysML in aerospace design and engineering," *Annals of Mathematics and Artificial Intelligence*, vol. 63, no. 1, pp. 53–102, Sep. 2011.
9. F. Ambert, F. Bouquet, J. Lasalle, B. Legeard, and F. Peureux, "Applying a def-use approach on signal exchange to implement SysML model-based testing," in 9^{th} *Europ. Conf. on Modeling Foundations and Applications (ECMFA'13)*, ser. LNCS, vol. 7949. Montpellier, France: Springer, Jul. 2013, pp. 134–151.
10. F. Bouquet, C. Grandpierre, B. Legeard, F. Peureux, N. Vacelet, and M. Utting, "A Subset of Precise UML for Model-based Testing," in 3^{rd} *Int. Workshop on Advances in Model-based Testing. (A-MOST'07)*. ACM, July 2007, pp. 95–104.
11. F. Bouquet, B. Legeard, and F. Peureux, "CLPS-B: A constraint solver to animate a B specification," *International Journal on Software Tools for Technology Transfer, STTT*, vol. 6, no. 2, pp. 143–157, Aug. 2004.
12. F. Ambert, F. Bouquet, S. Chemin, S. Guenaud, B. Legeard, F. Peureux, N. Vacelet, and M. Utting, "BZ-TT: A Tool-Set for Test Generation from Z and B using Constraint Logic Programming," in *Formal Approaches to Testing of Software (FATES'02)*. Rob Hierons and Thierry Jéron, August 2002, pp. 105–120.
13. J. Lasalle, F. Peureux, and F. Fondement, "Development of an automated MBT toolchain from UML/SysML models," *Innovations in Systems and Software Engineering*, vol. 7, no. 4, pp. 247–256, Dec. 2011.
14. A. K. Macworth, "Consistency in networks of relations," *Journal of Artificial Intelligence*, vol. 8, no. 1, pp. 99–118, Feb. 1977.
15. J. Gauthier, F. Bouquet, A. Hammad, and F. Peureux, "Tooled process for early validation of SysML models using Modelica simulation," in 6^{th} *Int. Conf. on Fundamentals of Software Engineering (FSEN'15)*, Tehran, Iran, Apr. 2015.
16. B. P. Zeigler, H. Praehofer, and T. Kim, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, 2nd ed. Elsevier Science, 2000, ISBN: 0-12-778455-1.
17. E. Lee, "Constructive Models of Discrete and Continuous Physical Phenomena," *Access, IEEE*, vol. 2, pp. 797–821, Aug. 2014.
18. M. Iqbal, A. Arcuri, and L. Briand, "Environment modeling and simulation for automated testing of soft real-time embedded software," *Software and System Modeling*, vol. 14, no. 1, pp. 483–524, Feb. 2015.
19. W. E. McUmber and B. H. C. Cheng, "UML-based analysis of embedded systems using a mapping to VHDL," in 4^{th} *Int. Symposium on High-Assurance Systems Engineering (HASE'99)*. Washington, DC, USA: IEEE CS, Nov. 1999, pp. 56–63.
20. Y. Vanderperren and W. Dehaene, "From UML/SysML to Matlab/Simulink: Current state and future perspectives," in 9^{th} *Int. Conf. on Design, Automation and Test in Europe (DATE'06)*. Munich, Germany: EDAA, Mar. 2006, pp. 93–93.
21. F. Boutekkouk, "Automatic SystemC code generation from UML models at early stages of systems on chip design," *Int. Journal of Computer Applications*, vol. 8, no. 6, pp. 10–17, 2010.