JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 13, No. 1, March 2019, pp. 17-31

SimulML: A DSML for Simulating SysML Models

Bassim Chabibi*, Mahmoud Nassar
IT Architecture and Model Driven Systems Development (IMS), ADMIR Laboratory, Rabat IT Center, ENSIAS, Mohammed V University in Rabat, Morocco
bassim.chabibi@um5s.net.ma, m.nassar@um5s.net.ma

Adil Anwar
SIWEB, E3S, EMI, Mohammed V University in Rabat, Morocco
anwar@emi.ac.ma

Abstract
SysML language offers strong capabilities to specify, analyze and design complex systems. However, the operational semantics of the diagrams is not precisely defined, which affects the implementation of SysML models. Furthermore, the SysML models are insufficient for system verification addressing various stakeholders' requirements. This aspect is managed by the simulation process. As system modeling and simulation analysis are used during the system development process, its efficiency is considerably reduced. This study integrates SysML modeling and simulation, through a defined intermediate domain-specific language, namely Simulation Modeling Language, which is intended to link SysML and simulation environments. The defined intermediate modeling language permits modeling complex systems via specification of its syntax and semantics, using simulation principles and constructs, in addition to bridging the gap between SysML models and different simulation environments. Utilizing the defined intermediate language, the proposed integration approach is facilitated by the potential and strength of both SysML modeling and simulation. Moreover, this integration approach is based on the concepts of Model-Driven Engineering ensuring an efficient passage between different models.

Category: Compilers / Programming Languages

Keywords: System engineering; MDE; SysML; Simulation; DSL; Model transformation

I. INTRODUCTION

A. Model-Driven Engineering

Model-Driven Engineering (MDE) is considered as a form of generative engineering in which models occupy a prominent place among system development artefacts, facilitating them partially or fully [1]. Indeed, MDE encourages the use of a large number of models to describe separately each of the concerns of users, designers, architects, etc. As a matter of fact, the notion of model is considered central to the concept of MDE.

In order to create a productive model, it must allow manipulation by a machine. For this purpose, the language with which this model is described must be clearly defined. Therefore, the definition of modeling language has taken the form of a model, called metamodel. A metamodel can be defined as a specification model for which the studied system is considered as a model in a certain modeling language [2]. In other words, a metamodel

Open Access http://dx.doi.org/10.5626/JCSE.2019.13.1.17 http://jcse.kiise.org

Received 25 December 2017; **Revised** 28 January 2019; **Accepted** 17 February 2019
*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

is a language model that describes its concepts and properties, its textual and/or graphic syntax, and its semantics.

Furthermore, the definition of domain-specific languages (DSL) is promoted by MDE. The reason is that DSLs are suitable for particular corporations or needs [3]. In addition, the Object Management Group (OMG) has defined a new orientation in 2000, namely Model-Driven Architecture (MDA), in order to handle the evolution and construction problems of recent systems. The main aim of this approach is its high level abstraction positioning, and focus on model approaches [4]. To this end, the fundamental principle of MDA includes generating platform-specific models (PSMs) from platform-independent models (PIMs). This action is fulfilled through various MDE techniques, as model transformation.

The latter is defined as an operation that automatically creates a set of target models from a set of source models [5]. Depending on the nature of the source and target metamodels, a transformation may be endogenous or exogenous. Thus, a transformation is considered endogenous if the target and source models conform to the same metamodel. Otherwise, it is considered as exogenous.

Model transformation languages can be broadly categorized into two categories, depending on the nature of data production: model-to-model (M2M) and model-to-text (M2T) transformations. In this context, various model transformation languages are included according to the two aforementioned categories, such as ATL (M2M) and Acceleo (M2T).

- **ATL:** This transformation language is part of the QVT transformation languages family. It consists of several QVT features; however, it lacks a few, such as bi-directionality, update transformations, etc.
  ATL is used in the context of a three-level architecture. Inspired by Jouault et al. [6], Fig. 1 illustrates the transformation mechanism from an M_a model to an M_b model controlled by a transformation program mma2mmb.atl, written in ATL. Moreover, the models M_a, M_b, as well as the transformation program mma2mmb.atl, conform to the metamodels MM_a, MM_b and ATL, respectively, which conform to the metamodel MOF.
- **Acceleo:** This language is an Eclipse implementation of the OMG MOF2Text transformation language (MTF) [7]. Acceleo is considered as a code generation language, facilitating generation of a structured file from an Eclipse modeling framework (EMF) model. This generated file consists of a text that is a programming language or another formal component. In this context, Acceleo requires the definition of an EMF metamodel and a model that conforms to the metamodel generating the text [8].
  The Acceleo language consists of a template-based approach. Templates are textual models containing spaces reserved for data extraction from the source models. These placeholders are expressions specified through the metamodel entities, defining queries constituting the primary mechanisms for the selection and extraction of source model values. The latter are then converted into text fragments through predefined libraries.

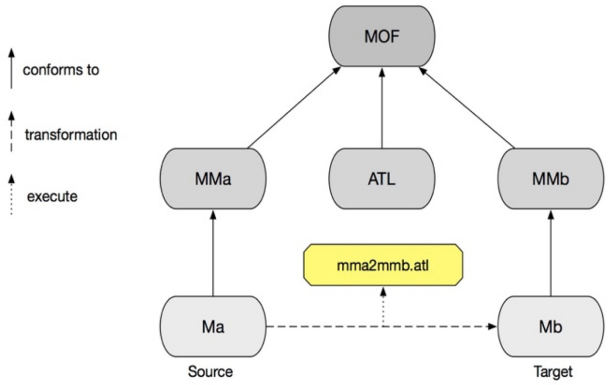

Fig. 1. ATL architecture.

B. System Engineering and SysML

System engineering is a multidisciplinary scientific approach aimed at developing solutions in response to the requirements of different stakeholders [9]. It is a general methodological approach that includes all the appropriate activities to design, develop and verify a system that provides a cost-effective and efficient solution to client needs, in addition to satisfying all stakeholders' requirements [10]. System engineering describes a structured development process, from the design to the operational phase. It takes into consideration technical and economic aspects to ensure the development of a system that meets users' needs.

Constrained by multiple stakeholder goals and large amounts of design information, system engineering projects are becoming more complex. To assist engineers cope with this growing complexity, a model-oriented approach has been adopted. As a result, model-based systems engineering (MBSE) has become an accepted approach for the design of complex systems because it solves system engineering problems by expressing multiple views that can transform the requirements of the parties and stakeholders via detailed specification using templates [11].

Considered as a language for graphic description of systems including combinations of hardware, software, data, etc., SysML [12] supports the analysis, specification, design, verification and validation of complex systems. The SysML language facilitates the MBSE approach via design of a cohesive and consistent model of the system. Nevertheless, SysML is not considered a method but rather as a set of graphical tools defined by a metalanguage providing the designer with all the facilities for model

construction with strong semantic coherence [13].

SysML is basically a UML [14] profile for modeling a broad spectrum of systems. As a result, their structures and concepts remain broadly similar. Nevertheless, differences exist involving some new stereotypes and diagrams. Moreover, the major difference between the two languages is the adoption of the "block" principle in the case of SysML, instead of the "class" in the case of UML. Indeed, if the class is the basic building block of the UML language, the block is considered as the modular unit of the SysML structure describing a type of system, a component, or an element that circulates through the system.

In addition to adopted and sometimes modified diagrams of those of UML 2.0, the requirement and parametric diagrams (PD) are new features. The first one allows specification of the application needs. As for the second, its role is to specify mathematical expression between the elements of the models. The block definition diagram (BDD) and the internal block diagram (IBD) were obtained by profound modification of the UML class diagram and composite diagram.

Even though SysML offers strong capabilities to specify, analyze and design complex systems including hardware, software, human, procedural and resource aspects, the operational semantics of the diagrams are not precisely defined, which constitutes an obstacle to the execution of SysML models. In addition, some systems (such as embedded systems) are described by a combination of event-driven models and continuous-time models. While SysML describes event-driven behavior, thanks to the inherited capabilities of UML, it does not facilitate the description of continuous-time behavior [15]. This aspect is well handled by the simulation process.

C. Combination between SysML Modeling and Simulation

Simulation is one of the most common techniques that use models to answer questions about systems and their behavioral changes over time. A simulation model of a system is defined as any model in which an experiment (process of extracting information from a system by stimulating its inputs) can be applied, to address questions about the system [16]. Depending on how the model is represented, it can be classified into mental, verbal, physical, and mathematical types. The latter is a description of a system whose variables are linked mathematically.

The frequent use of the simulation process is related, in particular, to the availability of various highly-specific simulation environments, and the development and optimization of simulation modeling methodologies. In addition, simulation is very useful and efficient for several reasons [17]. In this context, simulation allows the study and experimentation of the internal interactions of a complex system or one of its subsystems. Moreover, the information obtained through the design of the simulation model can be very useful in proposing system improvements in the study phase.

Among the various simulation environments that have been developed, Modelica [18] is considered as a declarative, object-oriented, acausal and equation-based language. Modelica is particularly suited to the modeling and simulation of mechatronic, electronic, hydraulic and aerodynamic systems. Furthermore, Modelica is not a tool but a language specification. Several tools are available in this context for implementation of this specification, such as OpenModelica and Dymola.

Modelica programs are built around the notion of "class", also called "models". The definition of a class allows the creation of a type, whose elements (or instances) exhibit the same properties as the parent class. A Modelica class contains a number of elements. The basic elements include variables and equations, the former contain data about instances of the class and contribute to preservation of instance data. As for the equations, they specify behavior of class instances. Moreover, the Modelica parameter keyword indicates that it is a variable whose value can be modified at the beginning but not later during the simulation. Variables can be typed integer, real, boolean, string, complex, or enumeration.

Due to the object-oriented aspect of Modelica language, a model contains a set of objects, each with its own characteristics and dynamics. These objects can be connected. In this context, this connection is generated through a specific class named "Connector". The latter introduces two types of variables, namely, non-flow variables (effort, potential, level) and flow variables. In addition, the connection between two connectors generates equations linking the connector variables. As a result, the non-flow variables of these linked connectors are equal (Kirchhoff node law), while the sum of flux variables is zero (Kirchhoff mesh law).

Given the complementarity aspect between SysML modeling and simulation, several studies have focused on the combination of these two approaches, in order to take advantage of the potentials and capacities of each of them. In this context, we performed a literature review [19] of the taxonomy of simulation environments (as MATLAB/Simulink, SystemC, VHDL, Modelica) that were combined with SysML modeling. Continuing with this work, we proposed an integrated approach based on co-simulation of SysML modeling and MATLAB/Simulink simulation environment [20].

The combination of SysML and simulation is based on the analysis of the concepts of both approaches. Indeed, an efficient and formal integration of SysML and the different simulation environments depends on elucidation of their similarities and differences. In this context, we performed a comparative study of SysML and simulation constructs, reported by Chabibi et al. [19]. This study has

allowed definition of an intermediate DSL, based on the common constructs of several simulation environments as well as simulation modeling methodologies, in order to bridge the gap between SysML modeling and simulation environments. The intermediate language is designated as Simulation Modeling Language (SimulML).

II. MODELS TRANSFORMATION FROM SYSML TO MATLAB

A. The Proposed Integration Approach

Our integration approach is mainly based on an intermediate language, carrying the necessary syntax and semantics that combines SysML language with the various simulation environments. Known as SimulML, this language is based on common concepts, semantics and modeling methodologies of several simulation tools. In Chabibi et al. [21], we defined SimulML via specification of both its syntax and semantics. In addition, SimulML facilitates the specification of a system through common concepts and simulation modeling methodologies.

SimulML plays the role of an intermediate language between SysML and the various simulation environments. Fig. 2 illustrates our integration approach through this language. Indeed, the integration between SysML and the simulation environments is based on the specification of a first passage between SysML and SimulML languages, and a second passage between SimulML language and the target simulation environment.

The passage between SysML and SimulML languages, considered as PIMs, is dependent on the specification of a model transformation between them. Towards this end, a mapping between the concepts of the two languages is key to ensure an efficient and formal transformation. Nevertheless, SysML language lacks a few concepts, used in simulation, such as the specification of properties as variables and constants. To remedy this limitation,

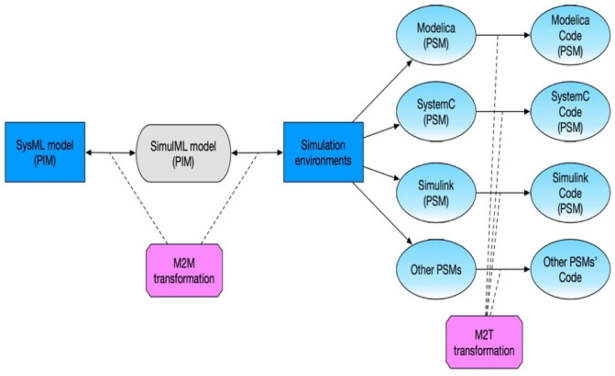

Fig. 2. Linking SysML modeling to simulation environments through SimulML.

SysML is enriched with specific stereotypes, aiming to optimize the model transformation process between this language and SimulML. Therefore, a SysML profile, known as SysML4Simulation, was defined in order to allow modeling of a complex system for transformation to a SimulML model. The model transformation between SysML and SimulML languages is qualified as an M2M transformation, since both languages are specified through their metamodels.

As for the transition between SimulML and simulation environments, it depends on the specification of a model transformation between SimulML concepts and those of the target simulation environment. Since the definition of SimulML is based, among other things, on common concepts of the most frequently used simulation environments, the mapping used in the specification of model transformation rules, relies on the establishment of the similarity between SimulML and the target simulation environment concepts. In this paper, we specify Modelica as the target simulation environment. In this context, we define a passage between SimulML and Modelica, considered as a PSM. This passage includes two transformations. The first one is an M2M transformation, used to generate Modelica models from SimulML models. As for the second one, it consists of an M2T transformation, whose objective is to generate Modelica code from the models for use in a Modelica environment, to simulate the studied models. These simulation models offer the possibility of simulating the studied system in order to verify its behavior and to study its feasibility criteria expressed by the stakeholders.

B. Simulation Modeling Language

The definition of SimulML is based on the specification of its abstract and concrete syntax, and its semantics. The abstract syntax relates to the internal representation of the model, manipulated by the computer, while the concrete syntax is related to the set of decorations (textual or graphic), as well as the connection between abstract syntax concepts and their corresponding decorations in concrete syntax. As for semantics, it facilitates precise and unambiguous definition of the concepts in the abstract syntax.

1) SimulML Abstract Syntax

The Eclipse-EMF/Ecore environment is used to define the abstract syntax of SimulML. The EMF is a modeling environment and code generation tool facilitating description of tools for other applications that are based on a structured data model. The EMF environment core contains a metamodel (Ecore) for description and support of model execution, including a notification change, persistence support with default XMI serialization, and an interface for reflective application programming (API) and generic manipulation of EMF objects.

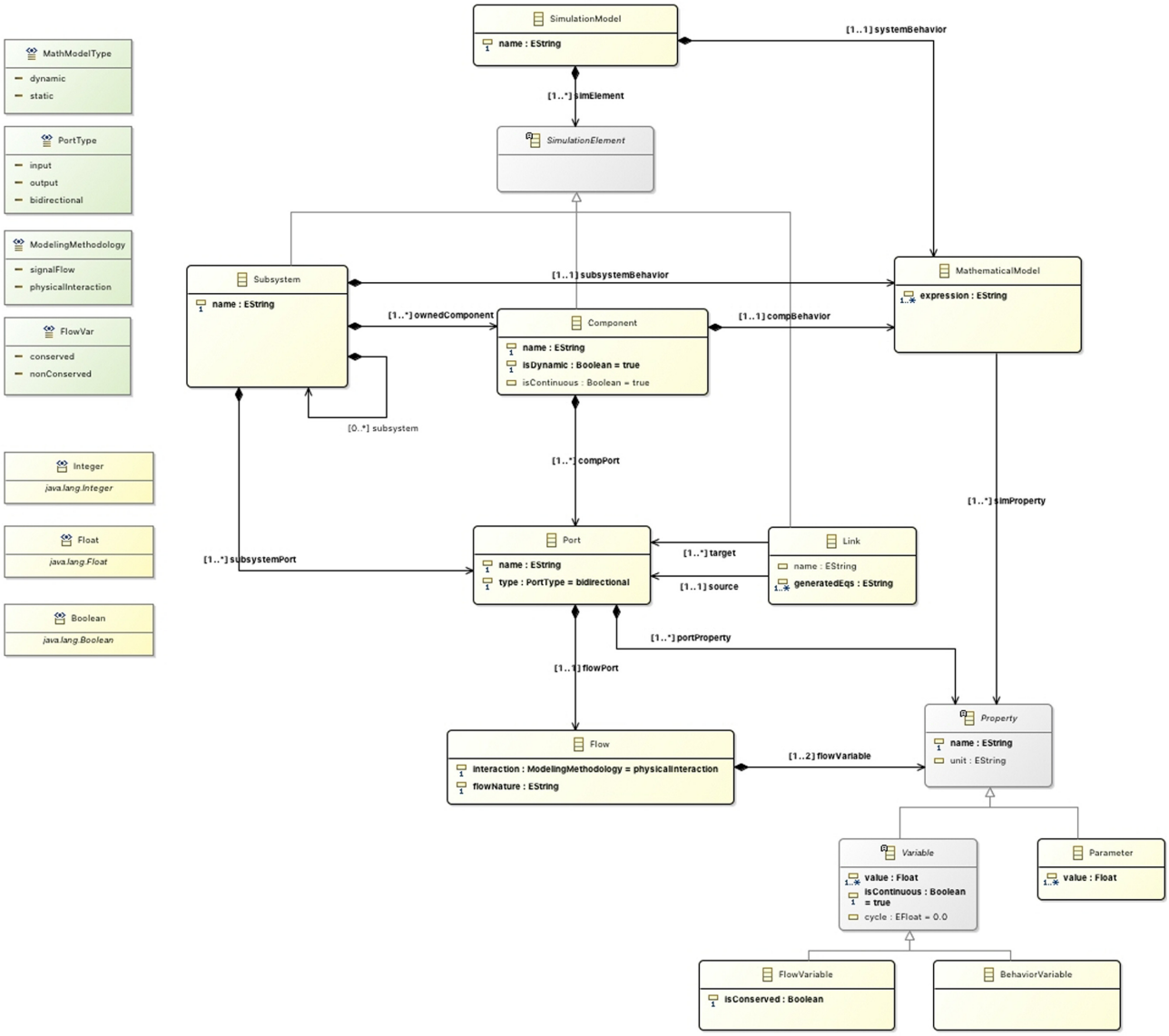

Fig. 3. SimulML abstract syntax.

Simulation modeling methodologies allow specification of information generated by simulation. In this context, the interaction between components depends on the exchange of information or physical entities. Therefore, two types of interaction between the components of a simulation model were specified in Matei and Bock [22] as follows:

- Physical interaction: Components exchange different kinds of physical substances;
- Signal flow: Components exchange digital information.

The SimulML metamodel is illustrated in Fig. 3. A simulation model is defined at the base of four main elements, as mentioned in Matei and Bock [23], as follows:

- **Component:** Considered as the atomic elements on which models are based, components consist of variables and parameters. Variables represent quantities that change over time, while parameters remain unchanged during the simulation. Variables and parameters are used in mathematical relationships (equations) to specify the mathematical model describing the component behavior.
- **Port:** It consists of interfaces used by components and subsystems to interact with other components and subsystems. This interaction is based on sharing or exchange of variables with other components or subsystems across these ports. In addition, the ports used in the case of signal-flow modeling are specified

as input or output ports, while those used in the case of physical interaction modeling, are considered as bidirectional.

- **Link:** In simulation, links between components represent only the mathematical models between their variables. They do not specify the flow physical nature unlike system engineering. In other words, simulation model links do not allow creation, destruction, transformation, backup, resistance or temporal variation of the flow of physical substances [24]. A connection between two components depends on the connection between their ports through links. These allow the exchange of energy or information between components. Once the connection is established between components via links, mathematical formulae are generated, which become part of the behavior of the whole model.
- **Subsystem:** It is a graphic concept containing components and/or subsystems. It aims to simplify the representation of a set of components and/or subsystems. A subsystem contains ports that interact with the other ports through links. In addition, the behavior of a subsystem is specified through the behavior of its elements.

The specification of SimulML abstract syntax allows specification of its basic concepts through their consistent representation, in addition to relationships that exist between them. However, the abstract syntax remains insufficient to clarify the model and interpret it unambiguously. For this purpose, it is essential to define SimulML semantics.

2) SimulML Semantics

In order to better understand and manipulate SimulML language, some semantics are expressed directly involving the metamodel, based on the notion of multiplicities. For example, the behavior of a component, subsystem or simulation model is described by means of a mathematical model. Each of these entities must have a single model containing one or more mathematical expressions that link the properties of different ports. Therefore, the constraint stipulating that a component, subsystem or simulation model must contain a mathematical model is specified through the relationship-specific "Composition", linking "Component" and "MathematicalModel" superclasses of the SimulML metamodel, as illustrated in Fig. 3. This composition has a cardinality (1..1), reflecting the uniqueness of the mathematical model vis-a-vis these entities.

Moreover, in order to define specific properties in the SimulML abstract syntax, the metamodel was supplemented by constraints expressed in OCL. The latter is a functional language based on first-order logic for expressing specifications of a UML model. OCL can be used as a specification language for metamodels, which requires adequate knowledge of the syntax and semantics of the modeling elements. In addition, well-designed rules can be added to metamodels to restrict all valid instances.

As a fact, all constraints expressed via OCL in the SimulML metamodel aim to clarify the meaning of concepts defined in abstract syntax and to remove any ambiguity concerning their use. In addition, in order to manipulate SimulML abstract syntax and generate its applications, it is imperative to specify its concrete syntax.

3) SimulML Concrete Syntax

The specification of the concrete syntax can be realized via several environments. In the case of SimulML, its concrete syntax is defined via Xtext. The latter is an environment for programming languages and DSML development. The Xtext environment consists of ANTLR (ANother Tool for Language Recognition)-generated grammar, as well as the EMF modeling environment. The Xtext covers all aspects of an integrated development environment (IDE): parser, compiler, interpreter, and full integration into the Eclipse development environment.

The description of SimulML concrete syntax was detailed in previous studies [21, 25]. Furthermore, concrete syntax was not used in this paper, as we aim to perform model transformations of several languages involved in our integrated approach. As a matter of fact, the specification of different model transformations relies mainly on involved languages metamodels.

C. Models Transformations

1) Passage from SysML to SimulML

If SysML is considered as a language that offers a variety of easy-to-use and effective concepts for modeling a wide range of system engineering issues, SimulML is performed on the basis of common concepts, semantics, and modeling methodologies used by several simulation environments. In order to establish a model transformation between these two languages, a mapping between their basic concepts is essential. In this context, regarding the multiple similarities between SysML structural diagrams and common concepts of most used simulation environments detailed in Chabibi et al. [21], the proposed model transformations are based on the use of these SysML diagrams.

However, there are some differences between SysML structural diagrams and simulation models [24]. The main differences are as follows:

- **Properties value specification:** In simulation, components consist of variables and parameters. If variables vary with time, parameters remain unchangeable during the simulation. In this context, property values (variables or constants) can be specified by several simulation environments. However, SysML does not provide such a specification. In addition, variables are designated, in simulation, as continuous or discrete. Continuous variables are entities changing values

continuously with time, while discrete variables change during well-defined time intervals. However, this specification is not provided by SysML.

- **Flow specification:** In SysML structural diagrams, the connection between blocks is specified through connectors that link their ports. The latter exhibit flow properties, indicating flow direction, as well as the nature of substances that constitute the flow. Nevertheless, SysML flow properties do not permit to specify neither flow rate nor potential to flow.
- **Link semantics' specification:** In simulation, linking components generate mathematical equation that contribute to system behavior. The generated mathematical equations consist of relationships between conserved and non-conserved variables, depending on the modeling methodology adopted [21]. Even though SysML connectors provide adequate semantics to represent equalities between the non-conserved variables of a link, they do not allow linking of conserved variables.

In order to develop model transformations between SysML and SimulML languages, it is necessary to enrich SysML constructs with new stereotypes. The aim is to overcome SysML lacking in simulation semantics. As a consequence, we defined the SysML4Simulation profile.

2) SysML4Simulation Profile

SysML has been enriched with four stereotypes as follows:

- **SmlSubsystem:** The construct SimulML subsystem refers to an entity that contains components and/or subsystems. It corresponds to a SysML block with internal structure, while an atomic component is compared to a SysML block without internal structure. In this context, the SmlSubsystem stereotype is created as a SysML block with an internal structure since it allows encapsulation of other blocks. The abstract syntax of the SmlSubsystem stereotype is illustrated in Fig. 4.
- **SmlConstant:** SysML does not allow specification of a property during a simulation execution, similar to SimulML. For this purpose, the SmlConstant stereotype is defined. It is a stereotype extending the Property metaclass, as shown in Fig. 5. It should be noted that the word "parameter" was not retained to designate this stereotype, since it plays a significant role in SysML, which differs from that of SimulML.
- **SmlVariable:** Extending the Property metaclass, this stereotype allows representation of a property that changes with time during the simulation execution, as well as specification of continuous or discrete nature. For this purpose, as shown in Fig. 6, SmlVariable stereotype has two attributes. The first one, named *isContinuous*, indicates whether the variable is continuous or discrete, through a Boolean value. Its default value is "true". As for the second attribute, named *cycle*, it specifies the cycle after which a discrete variable (isContinuous = false) must change its value, via a real value. As a result, a variable

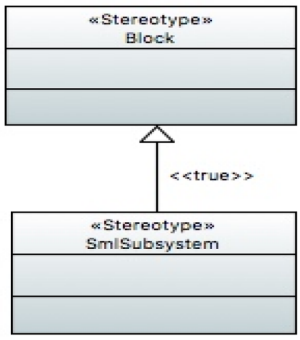

Fig. 4. SmlSubsystem stereotype.

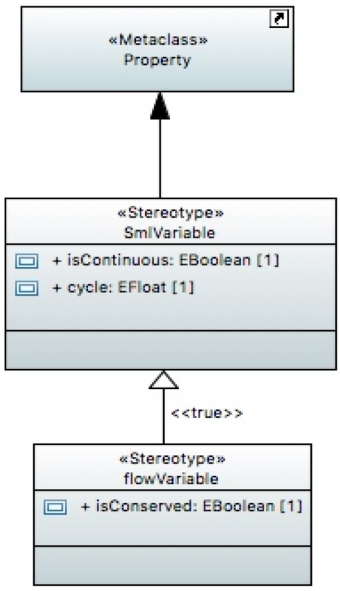

Fig. 5. SmlConstant stereotype.

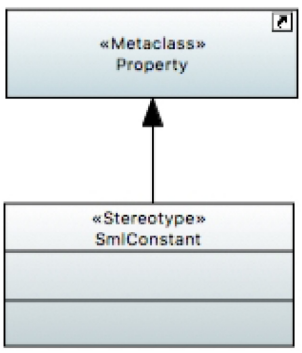

Fig. 6. SmlVariable and flowVariable stereotypes.

remains constant during the cycle period. In the case of a continuous variable (isContinuous = true), the value of "cycle" is equal to zero, meaning that the variable varies continuously.

- **flowVariable:** In the simulation, linking components and/or subsystems generates semantics of flow properties including flow rate and potential to flow. These are not specified by SysML language. For this purpose, the flowVariable stereotype, extending the Property metaclass, is defined. As shown in Fig. 5, this stereotype has a Boolean value (its default value is false), named *isConserved*, in addition to the attributes inherited from the SmlVariable stereotype. The aim of this attribute is to distinguish between a conserved variable (flow rate) and a non-conserved variable (potential to flow).

Thus, SysML concepts are enriched by four stereotypes in order to fill the gaps concerning its differences with SimulML. The created SysML profile is used in the definition of model transformations, through ATL language, in order to combine both SysML and SimulML languages.

3) SysML2Simulation Transformation

The definition of a model transformation depends on manipulation of constructs in both source and target metamodels, as shown in Fig. 1. Therefore, implementing model transformation requires a good knowledge of metamodels, as well as the link between their constructs. Indeed, accurate specification of mapping between source and target metamodels constructs is necessary for definition of the transformation rules.

In this context, mapping between SysML and SimulML constructs is specified on the basis of previous works [21]. Table 1 summarizes the mapping between the constructs of these two languages.

Model transformation rules are specified via ATL language. As a fact, SysML2SimulML transformation is based on an ATL module that includes the definition of a header, and a set of functions and transformation rules. ATL functions are also called *Helpers,* and are classified into two categories: helper attribute and helper operation. As for ATL transformation rules, they serve to express several types of transformation logic. Fig. 7 illustrates an ATL rule expressed in SysML2SimulML transformation module. It stipulates that a SysML block transformation is subject to specific constraints as follows:

- If a SysML block is not abstract, it is transformed into an atomic SimulML component;
- If a SysML block is not abstract, but stereotyped "SmlSubsystem", it is transformed into a SimulML subsystem;
- If a SysML block is abstract and its name ends with "_FlowElements", it is saved in "allAbstractBlocks" sequence, in order to generate its port, its flow and its flow variables.

Through the SysML2SimulML model transformation, SimulML models are generated conforming to source SysML models. The generated SimulML models are built according to principles and constraints defined in SimuML abstract syntax. Moreover, SimulML models are used as intermediate entities to produce Modelica models. Furthermore, it should be noted that the generated SimulML models represent the modeling of source SysML models

Table 1. Mapping between SysML and SimulML constructs

SimulML components	SysML
Simulation model	Block specified as an IBD
Component	Atomic block
Subsystem	SmlSubsystem stereotype
Port	Flow port
Link	Connector
Mathematical model	Constraint block
Flow	Flow specification
Parameter	SmlConstant stereotype
Behavioral variable	SmlVariable stereotype
Flow variable	flowVariable stereotype

```
--Transformation of SysML Blocks
rule block2SimulationElement {
    from block : SYSML!Block

    do {
        if(block.base_Class.getAppliedStereotypes()->collect(e | e.name).includes('SmlSubsystem'))
            thisModule.createSubsystem(block);

        else
            if(not block.base_Class.isAbstract and block.base_Class.getAppliedStereotypes()->collect(e | e.name).includes('Block'))
                thisModule.createComponent(block);

            if(block.base_Class.isAbstract and block.base_Class.name.endsWith('_FlowElements'))
                -- Save this abstract block in a sequence
                thisModule.allAbstractBlocks<-Sequence{thisModule.allAbstractBlocks, block}->flatten();
    }
}
```

Fig. 7. A transformation rule of SysML2SimulML transformation module.

through the basic constructs, semantics and modeling methodologies of simulation.

4) Passage from SimulML to Modelica

The transition between SimulML and Modelica depends on the specification of two model transformations (see Fig. 2). The first one is an M2M transformation between SimulML and Modelica metamodels. The second one is an M2T transformation that transforms Modelica-generated models into executable files (.mo), in order to use them in a Modelica simulation environment.

a) M2M transformation:

As SysML2SimulML transformation, SimulML2Modelica transformation is defined via ATL language, on the basis of mapping detailed in Table 2. SimulML2Modelica transformation depends on the specification of an ATL module, defining a set of functions and transformation rules, for efficient mapping between the basic constructs of both SimulML and Modelica languages defined by their correspondent metamodels.

An ATL module is based on the specification of a set of transformation rules. In this context, Fig. 8 describes a rule that transforms a SimulML subsystem into a Modelica model. In addition to saving the created model in a sequence named "AllModels", the rule states that the different components included in the transformed subsystem are used as parameters of the "createPartialClass" function. The latter is designed to generate partial classes, conforming to subsystem components, included in the Modelica model.

As a result, SimulML2Modelica module generates Modelica models that conform to source SimulML models. In order to simulate the generated models, an M2T transformation is defined in order to produce Modelica code from generated models. This M2T transformation is specified through the Acceleo language.

Table 2. Mapping between SimulML and Modelica constructs

SimulML concepts	Modelica
Component	Model (without connections)
Subsystem	Model (with connections)
Port	Connector
Link	Connection
Mathematical model::expression	Equation class
Flow	Not available
Flow direction	Causality of connection instance
Parameter	Parameter
Behavioral variable	Variable
Flow variable	
isConserved = true	Flow variable
isConserved = false	Variable

```
rule SmlSubsystem2Class {
    from
        ss : SimulML!Subsystem

    to
        model : Modelica!Model (name <- ss.name)

    do{
        -- Save this model in a sequence
        thisModule.allModels<-Sequence{thisModule.allModels, model}->flatten();

        --Search for ownedComponents
        for(c in ss.ownedComponent){
            thisModule.createPartialClass(c, model);
        }
    }
}
```

Fig. 8. An example of a transformation rule in SimulML2Modelica module.

b) M2T transformation:

An M2T transformation is specified based on the perspective of generating Modelica code from the models produced via the SimulML2Modelica transformation. M2M transformation is defined through the Acceleo language. The main elements underlying this transformation are shown in Fig. 9. The M2T approach principle to facilitate the use of Acceleo models (.mtl) to define the transformation rules applied to Modelica source models. Acceleo models and Modelica source models are sent to an Eclipse generator, in order to generate the corresponding Modelica source code (.mo), conforming to the defined transformation rules.

Therefore, using the M2T transformation applied to Modelica models, an executable code is generated to produce simulations. The latter allow study and verification of system behavior, previously modeled via SysML language. In this context, the generated Modelica code is used to produce simulations by exploiting the Eclipse Modelica Development Tooling plugin (MDT) [26]. The latter incorporates *OpenModelica Compiler*, which provides Modelica simulation models.

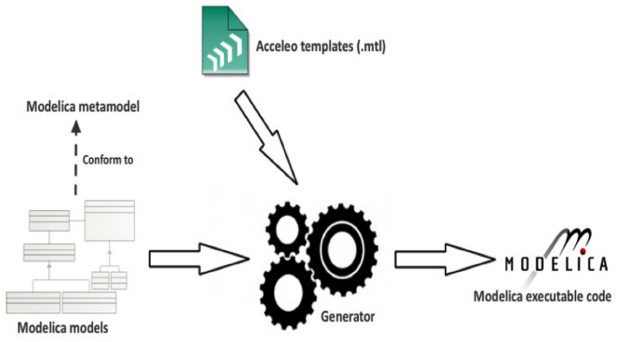

Fig. 9. Modelica code generator.

In order to illustrate the efficiency of this integration, the following chapter highlights a case study involving electrical circuit modeling. The goal is to conduct verifications through simulations. For this purpose, the case system was modeled via SysML4SimulML profile. The SysML model undergoes various transformations specified in order to yield an executable Modelica code, facilitating experiments in the study system.

III. AN ILLUSTRATIVE EXAMPLE

In order to illustrate the different stages of our integration approach, we propose the study of a system that consists of an electrical circuit integrating two main components, namely, a "chopper" and a "DC motor". The objective of this study is to simulate the behavior of the direct current (DC) motor in order to observe and control its rotational speed, according to the constant voltage of a power source.

For this purpose, this case study aims first at modeling the system through the SysML4Simulation profile. The created SysML models are transformed, step by step, in order to develop an executable Modelica code, allowing simulations designed to validate predefined constraints.

A. The Case Study Description

The system used to illustrate the efficiency of our integration approach has been used in Chabibi et al. [27], in order to highlight a model integration approach that links SysML to MATLAB/Simulink simulation environment. Therefore, we explored the same system as a case study in the proposed integration approach.

Fig. 10 illustrates the electrical circuit modeled. It consists on a source of electricity (i.e., battery), a chopper and a DC motor. The main aim of this study is to simulate the DC motor behavior in order to observe and control its rotation speed according to the constant DC voltage source.

The two main components of this electrical circuit are the chopper and the DC motor. The former is a device that converts fixed direct current input to a variable direct current output voltage directly. It is an electronic switch that chops a DC voltage in a more or less wide portion, which allows slot-shaped voltage and an adjustable average value derived from a fixed voltage, with a yield close to 1.

Operating in switching mode, the chopper switch is either opened or closed, according to the period T. The duration during which the chopper switch is closed is called t_F. Thus, the cyclical report a can be defined by the following equation:

$$\alpha = \frac{t_F}{T} \text{ with } 0 \le \alpha \le 1.$$

There are two functional modes of the chopper, according to the switching mode of the chopper switch, including the following:

- If $0 \le t < t_F$, the chopper switch is closed: $V_s = E$;
- If $t_F \le t < T$, the chopper switch is opened: $V_s = 0$.

As a fact, the voltage average value at the terminals of the DC motor is:

$$Vs_{moy} = \alpha.E \quad (1)$$

As for the DC motor, it is considered as an electro-mechanical converter that allows bidirectional conversion of energy between an electrical installation traversed by a direct current and a mechanical device. In fact, in the case of motor operation, the electrical energy is transformed into mechanical energy, and vice versa in the case of generator operation. Fig. 11 depicts some of the requirements for the DC motor operation. In the following study, we focus on the verification of the rotor rotation criteria stipulating that the rotor rotational speed must increase continuously according to DC output voltage of the chopper.

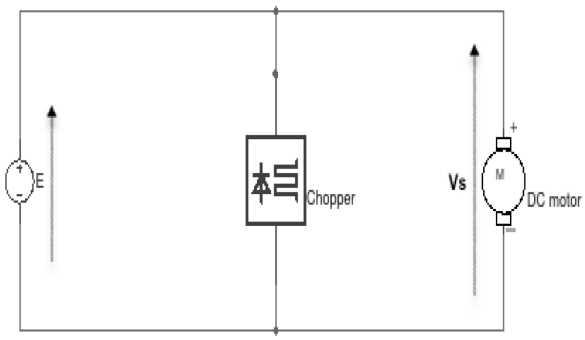

Fig. 10. Schematic of the studied electrical circuit.

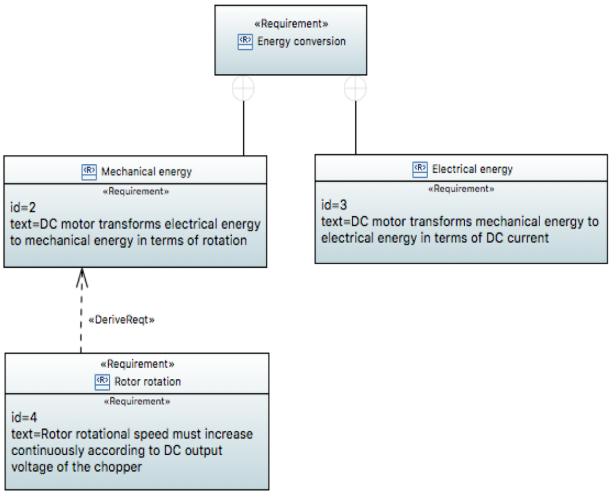

Fig. 11. Requirement diagram of the DC motor.

All electric motors are based on the physical principle of magnetic coupling between two magnetic fields. Electrical energy is transformed into mechanical energy through this magnetic coupling or interaction. Therefore, each motor carries two magnetic circuits, known as stator (fixed part) and rotor (moving part). In the case of a DC motor, the stator, also called inductor, creates a magnetic field; the rotor, also called armature, is fed by a direct current.

The key entity of the DC motor is the counter-electromotive force E_{cef}. It is specified through the following relation:

$$E_{cef} = K.n \tag{2}$$

K is a constructor data, which is generally called constant of speed, for low power motors, and n is the rotor rotational speed (tr/s).

In the case of a continuous regime, the voltage at the terminals of the DC motor is specified as follows:

$$V_s = E_{cef} + R.I \cong E_{cef} \tag{3}$$

R is the total resistance (i.e., cable, brushes, commutator blade and armature winding). As a consequence, the rotor rotational speed can be expressed according to the constant DC voltage source as follows:

$$n = \frac{Vs_{moy}}{K} \tag{4}$$

$$n = \frac{\alpha.E}{K} \tag{5}$$

B. SysML4SimulML Modeling of the Studied System

To illustrate the transformation of SysML2SimulML models, the case study system is modeled using SysML structural diagrams. Thus, Fig. 12 shows the BDD that models the electrical circuit, which is performed via the SysML4SimulML profile. Indeed, considered as a subsystem containing other blocks, the entity "Electrical_circuit" is modeled through the stereotype "SmlSubsystem". It consists of three blocks, namely the battery, the chopper and the MCC.

The block representing the "chopper" exhibits an alpha property, modeled through the "SmlVariable" stereotype. This block contains a constraint property "ch_eq", specified via the constraint block "chopper_CB". The

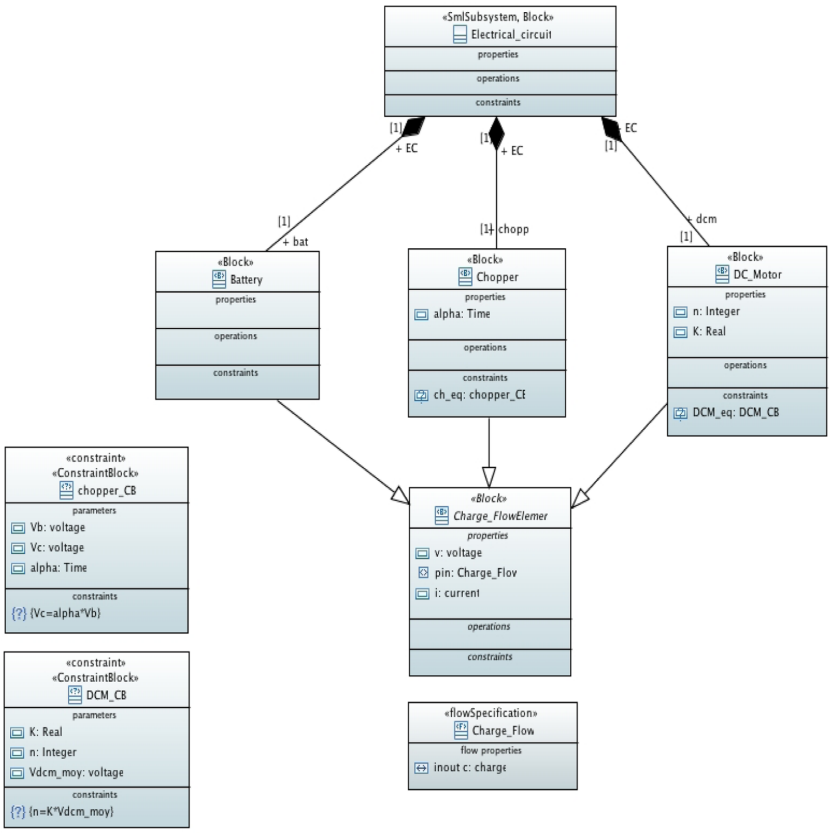

Fig. 12. The BDD of the electrical circuit.

latter yields the equation controlling the operation of the chopper (Eq. (1)), as well as the parameters thereof.

As for the block "DC_Motor", it contains the properties *n* and *K*, represented by the stereotype "SmlConstant". In addition, this block also expresses a constraint property "DCM_eq", specified through the constraint block "DCM_CB". The latter specifies the Eq. (5).

The flow exchange between different blocks includes electric charge. Since the modeling is based on physical interaction, flow elements are modeled through the abstract block "Charge_FlowElements". In addition to flow variables *v* and *i*, this abstract block contains the port "pin", whose specification is defined via "Charge_Flow". Thus, it appears that this port is bidirectional (inout), because of the exchange of physical substances (physical interaction modeling). Otherwise, the flow variables *v* and *i* are described as "flowVariable" stereotypes, specifying that the first variable is non-conserved, while the second variable is conserved. As the three blocks of the electric circuit exchange the same type of flow, the corresponding SysML blocks inherit the abstract block "Charge_FlowElements". As a consequence, they inherit the aforementioned flow variables and the "pin" port for the charge properties exchange.

Fig. 13 illustrates the IBD of the subsystem "Electrical_circuit". The IBD facilitates the flow based on electrical charge through the electrical circuit via connectors linking the flow ports. In addition, the connections between different ports facilitates the generation of flow variable equations, stipulating that the values of non-conserved variables are equal and the sum of the values of conserved variables is equal to zero.

As shown in Fig. 14, the parametric diagram of the block describing the DCM allows specific behavior, through the specification of Eq. (5), and its parameters. In this context, the equation describing the DCM behavior is described through the constraint property, defined in SysML constraint block. The latter insures reusability of system constraints. The parameters therein become the constrained properties.

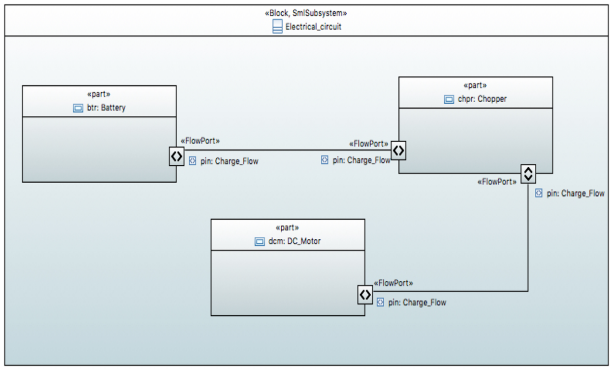

Fig. 13. An IBD of the electrical circuit.

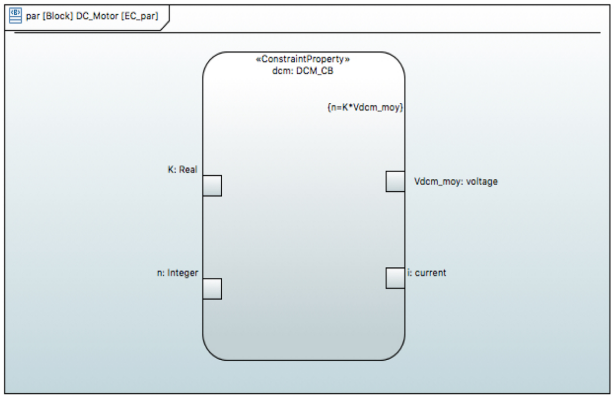

Fig. 14. Parametric diagram of the electrical circuit.

C. Model Transformations of System Models

1) Generation of SimulML Models through SysML2-SimulML Transformation

Based on the electrical circuit SysML models, the SimulML model was generated according to SysML2SimulML transformation. Fig. 15 illustrates the SimulML model generated in XML format. The SysML constructs used in the electrical circuit modeling were transformed into SimulML constructs, according to the mapping defined in Table 1. For example, the SmlSubsystem stereotype "Electrical_circuit" was transformed into a SimulML Subsystem. In addition, the different blocks constituting its parts were transformed into SimulML "ownedComponent".

Notably, the generated SimulML model reflects the studied system based on common constructs and modeling methodologies used in simulation. This representation is

```
<?xml version="1.0" encoding="UTF-8"?>
<simulML:SimulationModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:simulML="http://www.example.org/simulML">
  <simElement xsi:type="simulML:Subsystem" name="Electrical_circuit">
    <ownedComponent name="Battery">
      <compPort name="Battery:Charge_Flow:pin">
        <flowPort flowNature="charge">
          <flowVariable xsi:type="simulML:FlowVariable" name="charge_v" isConserved="false"/>
          <flowVariable xsi:type="simulML:FlowVariable" name="charge_i" isConserved="true"/>
        </flowPort>
      </compPort>
    </ownedComponent>
    <ownedComponent name="Chopper">
      <compBehavior simProperty="//@simElement.0/@ownedComponent.1/@compPort.0/@portProperty.0
      //@simElement.0/@ownedComponent.1/@compPort.0/@flowPort/@flowVariable.0
      //@simElement.0/@ownedComponent.1/@compPort.0/@portProperty.1">
        <expression>Vc=alpha*Vb</expression>
      </compBehavior>
      <compPort name="Chopper:Charge_Flow:pinB">
        <flowPort flowNature="charge">
          <flowVariable xsi:type="simulML:FlowVariable" name="charge_vb" isConserved="false"/>
          <flowVariable xsi:type="simulML:FlowVariable" name="charge_ib" isConserved="true"/>
        </flowPort>
        <portProperty xsi:type="simulML:Parameter" name="alpha"/>
        <portProperty xsi:type="simulML:BehaviorVariable" name="Vc"/>
      </compPort>
      <compPort name="Chopper:Charge_Flow:pinM">
        <flowPort flowNature="charge">
          <flowVariable xsi:type="simulML:FlowVariable" name="charge_vm" isConserved="false"/>
          <flowVariable xsi:type="simulML:FlowVariable" name="charge_im" isConserved="true"/>
        </flowPort>
      </compPort>
    </ownedComponent>
  </simElement>
</simulML:SimulationModel>
```

Fig. 15. The generated SimulML model (XML format).

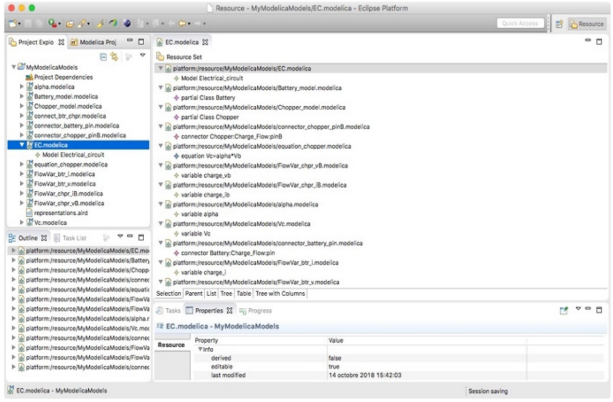

Fig. 16. The generated Modelica models through SimulML2Modelica transformation.

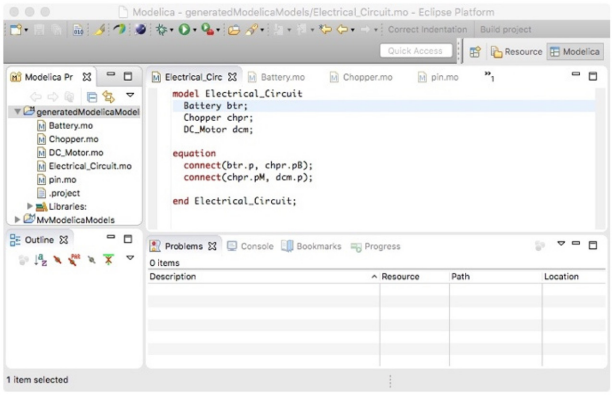

Fig. 17. Modelica script files (.mo) generated through M2T transformation.

an important contribution parameter of efficiency in the intermediate language SimulML.

2) Generation of Modelica Models through SimulML2Modelica Transformation

Fig. 16 presents the generated Modelica model. At the level of the project explorer (top left), we observe the different Modelica entities, generated from SimulML elements. Even though Modelica entities were generated separately, however, the links between them were specified through inclusion attributes "ownedClass".

Flow variables are included among the generated entities. These variables depend on the modeling methodology adopted. As long as the studied system was modeled via physical interaction, the "Charge" flow variables were divided into conserved (Current *i*) and non-conserved (Voltage *v*) variables.

SimulML's link construct was translated into Modelica's "Connect class", linking two Modelica connectors (ports in SimulML) and generating the resulting equations. In this context, the generated Connect class links the "pin" ports of both "Battery" and "Chopper" components. Based on these class properties, the linked connectors (btr.pin and chpr.pinB) were specified, as well as their containing class.

3) Modelica Code Generation and Simulation

Fig. 17 illustrates different script files (.mo), obtained from code generation performed with the Acceleo generator. These files represent Modelica classes of "Electrical_circuit", "Battery", "Chopper" and "DC_Motor", as well as "pin" connector. Connections between the system components were specified through Modelica Connect class. The latter allows generation of the connector variable equations.

Using the generated script files, a simulation can be performed to study the system behavior using the MDT console, embedded in Eclipse "OpenModelica Development Tooling" plugin. The latter also includes "OpenModelica Compiler" (OMC), which allows simulation.

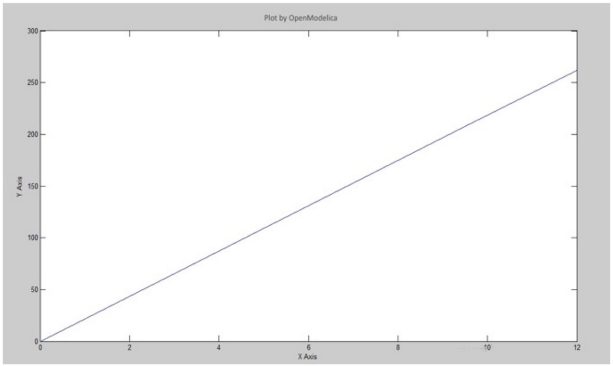

Fig. 18. DCM's rotation speed variation.

As indicated above, the purpose of this study was to simulate the behavior of the DC motor in order to observe and control its rotational speed as a function of the constant DC voltage source (in this case, the battery), as part of the verification of the rotor rotation criteria detailed in Fig. 11. For this purpose, we performed a simulation, based on the script files (.mo), in order to observe rotational speed variation (n) of the DC motor according to the chopper output voltage variable (Vs_{moy}). Since the source voltage E was equal to 12 V, and the speed constant K was estimated at 0.0458, the rotational speed of the rotor varied, as shown in Fig. 18. Therefore, we notice that the DCM rotation speed (n) increases continuously as a function of the chopper output voltage. Moreover, through this simulation, it is possible to conduct experiments that facilitate observation and study of the DCM speed of rotation, according to various input parameters (i.e., E, K, Vs_{moy}, etc.).

IV. DISCUSSION

This paper presents a model integration approach that

links SysML language to various simulation environments, using an intermediate language, namely SimulML. SimulML is based on a study of the main common constructs, semantics and modeling methodologies of simulation tools.

Thus, SimulML allows creation of a simulation model of a given system. This model consists of components and/or subsystems that are interconnected by links through ports. The behavior of a system is specified by the behavior of its components and their interaction. The behavior of a component is described through mathematical models based on component variables and parameters.

SimulML is used to transform SysML models into selected simulation environmental models. In this paper, we have bridged the gap between SysML and Modelica simulation language. To this end, the integration approach is based on M2M model transformations derived from SysML models to SimulML ones, and extended from the latter to Modelica. The Modelica models generated are transformed, via M2T transformation, into executable script files.

The SimulML is the major tool in our integration approach. SimulML represents our intermediate language linking SysML modeling to simulation environments (i.e., Modelica, etc.) through an approach that is compliant with MDE principles (i.e., metamodels, model transformations, etc.), as illustrated in Fig. 2, leveraging the potential offered by these two approaches.

REFERENCES

1. J. Bézivin and X. Blanc, "Promesses et interrogations de l'approche MDA," *Journal Développeur Référence*, pp. 1-14, 2002.
2. E. Seidewitz, "What models mean," *IEEE Software*, vol. 20, no. 5, pp. 26-32, 2003.
3. AS MDA, "Action Spécifique CNRS sur l'Ingénierie Dirigée par les Modèles," vol. 1.1.2, pp. 1-16, 2005.
4. J. Bezivin and X. Blanc, "MDA: vers un important changement de paradigme en génie logiciel," *Développeur Référence*, 2002; http://mfworld42.free.fr/cnam/nfe115-informatique_decisionnelle/MDA.Partie1.JBXB.Last.prn.pdf.
5. F. Jouault, "Contribution à l'étude des langages de transformation de modèles," Ph.D. dissertation, Université de Nantes, France, 2006.
6. F. Jouault, F. Allilaire, J. Bezivin, and I. Kurtev, "ATL: a model transformation tool," *Science of Computer Programming*, vol. 72, no. 1-2, pp. 31-39, 2008.
7. Object Management Group, "MOF Model to Text Transformation Language (MOFM2T) specification version 1.0," 2008; https://www.omg.org/spec/MOFM2T/About-MOFM2T/.
8. A. Abdulhameed, A. Hammad, H. Mountassir, and B. Tatibouet, "An approach combining simulation and verification for SysML using SystemC and Uppaal," in *Proceedings of the 8th French Conference on Software Architectures* (CAL), Paris, France, 2014.
9. S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. Waltham, MA: Morgan Kaufmann, 2012.
10. P. Roques, *Modélisation de Systèmes Complexes Avec SysML*. Paris, France: Eyrolles, 2013.
11. M. A. Abdul Rahman and M. Mizukawa, "Model-based development and simulation for robotic systems with SysML, Simulink and Simscape profiles," *International Journal of Advanced Robotic Systems*, vol. 10, article no. 112, 2013.
12. Object Management Group, "OMG System Modeling Language (SysML) specification version 1.3," 2012; https://www.omg.org/spec/SysML/1.3/About-SysML/.
13. J. P. Lamy, "SysML, un langage modèle," *Journal Technologie*, vol. 179, pp. 32-48, 2012.
14. Object Management Group, "UML Profile for MARTE specification version 1.0 beta," 2007; https://www.omg.org/spec/MARTE/1.0/Beta1/About-MARTE/.
15. R. Kawahara, D. Dotan, T. Sakairi, K. Ono, H. Nakamura, A. Kirshin, S. Hirose, and H. Ishikawa, "Verification of embedded system's specification using collaborative simulation of SysML and simulink models," in *Proceedings of 2009 International Conference on Model-Based Systems Engineering*, Haifa, Israel, 2009, pp. 21-28.
16. P. Fritzson, *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. Hoboken, NJ: John Wiley & Sons, 2011.
17. J. Banks, J. Carson, B. L. Nelson, and D. Nicol, *Discrete-Event System Simulation*, 4th ed. Upper Saddle River, NJ: Pearson, 2005.
18. Modelica Association, "Modelica - a unified object-oriented language for systems modeling (language specification version 3.3)," 2012; https://www.modelica.org/documents/ModelicaSpec33.pdf.
19. B. Chabibi, A. Anwar, and M. Nassar, "Towards an alignment of SysML and simulation tools," in *Proceedings of 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, Marrakech, Morocco, 2015, pp. 1-6.
20. B. Chabibi, A. Douche, A. Anwar, and M. Nassar, "Integrating SysML with simulation environments (simulink) by model transformation approach," in *Proceedings of 2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, Paris, France, 2016, pp. 148-150.
21. B. Chabibi, A. Anwar, and M. Nassar, "Metamodeling approach for creating an abstract representation of simulation tools concepts," in *Proceedings of 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, Agadir, Morocco, 2016, pp. 1-7.
22. I. Matei and C. Bock, *Modeling Methodologies and Simulation for Dynamical Systems*. Gaithersburg, MD: US Department of Commerce, National Institute of Standards and Technology, 2012.
23. I. Matei and C. Bock, "An analysis of solver-based simulation tools," National Institute of Standards and Technology, *Technical Report NISTIR 7846,* 2012.
24. C. Bock, R. Barbau, I. Matei, and M. Dadfarnia, "An extension of the systems modeling language for physical interaction and signal flow simulation," *Systems Engineering*,

vol. 20, no. 5, pp. 395-431, 2017.
25. B. Chabibi, A. Anwar, M. Nassar, L. Admir, and R. I. Center, "Model integration approach from SysML to MATLAB/ Simulink," *Journal of Digital Information Management*, vol. 16, no. 6, pp. 289-307, 2018.
26. E. Jagudin and A. Remar, "Modelica development tooling for eclipse," https://openmodelica.org/svn/MDT/trunk/docs/ Thesis.pdf.
27. B. Chabibi, A. Anwar, and M. Nassar, "Towards a model integration from SysML to MATLAB/Simulink," *Journal of Software*, vol. 13, no. 12, pp. 630-645, 2018.

Bassim Chabibi

Bassim Chabibi is a member of IT architecture and Model-driven Systems development (IMS) team, belonging to Advanced Digital Enterprise Modeling and Information Retrieval (ADMIR) laboratory of ENSIAS. He is a Ph.D. student in the domain of modeling and simulation of complex and embedded systems. In 2010, he obtained his engineering degree in Science computer and Automatic for Embedded Systems from ENSIETA School of Brest (France). Simultaneously, he received a Master degree in science computer research from UBO University of Brest (France).

Adil Anwar

Adil Anwar is currently an associate professor in computer science at the University of Mohammed-V in Rabat and a member of the Siweb research team of Mohammadia School of Engineers (Morocco). In 2009, he received a Ph.D. degree in Computer Science at the University of Toulouse (France). He is interested in software engineering, including model-driven software engineering, mainly by heterogeneous software language, traceability management in model-based systems engineering, combining formal and semi-formal methods in software and system's development.

Mahmoud Nassar

Mahmoud Nassar is a full professor at National Higher School for Computer Science and Systems Analysis (ENSIAS), Rabat, Morocco. He is Head of IMS (IT architecture and Model-Driven Systems development) Team / ADMIR Laboratory of Rabat IT Center. He received his Ph.D. in Computer Science from the INPT Institute of Toulouse, France. His research interests are in context-aware service-oriented computing, component based engineering, model-driven engineering, cloud computing, and cloud migration. He leads numerous R&D projects related to the application of these domains in smart cities, embedded systems, e-Health, and e-Tourism.