Towards a Model Integration from SysML to MATLAB/Simulink

Bassim Chabibi*, [1], Adil Anwar[2], Mahmoud Nassar[1]

[1] IT architecture and Model driven Systems development (IMS), Labo. ADMIR, Rabat IT Center, ENSIAS, Rabat Mohamed V University in RABAT, Morocco.
[2] SIWEB, E3S, EMI, Rabat Mohamed V University in RABAT, Morocco.

* Corresponding author. Tel.: +212 6 62 22 85 81; email: bassim.chabibi@um5s.net.ma
Manuscript submitted September 6, 2018; accepted October 30, 2018.
doi: 10.17706/jsw.13.12.630-645

Abstract: In system-level design, descriptive system models seem to be insufficient in order to perform a system verification which fulfils various stakeholders' requirements. This fact is accentuated by the increasing complexity of system engineering projects and, as a consequence, the difficulties to deal with both their coordination and traceability. Even if SysML (System Modeling Language) is considered as a flexible and standard tool for system engineering, using only descriptive models are insufficient for system behavior verifications. To deal with this concern, simulation environments (i.e. MATLAB/Simulink) allow verifying if the system preliminary design satisfies requirements or not. As a consequence, various research works have been centered on combining the potential of both SysML modeling and simulation tools. This paper proposes an integration approach based on metamodeling and model transformations to generate Simulink models from SysML diagrams. This approach is handled by models and modern techniques of MDE (Model-Driven Engineering).

Key words: MDE, model transformation, profiles, Simulation, simulink, SysML, system engineering.

1. Introduction

A System consists of a set of elements that mutually interact with one an another and can be viewed as a whole that interact with its external environment [1]. It can also be seen as an artifact created by humans that consists of components or blocks that pursue a common goal that cannot be achieved by each of the single elements [2]. Systems engineering (SE) is a multidisciplinary scientific approach aimed to develop solutions in response to the requirements of different stakeholders [1]. SE describes a structured development process, from design to operation phase by integrating several disciplines. It takes into account the technical and economic aspects to ensure the development of a system that meets the needs of users.

Constrained by the objectives of multiple stakeholders and the large quantities of design information, system engineering projects are becoming more and more complex. In order to help engineers to deal with this increasing complexity, a model-based approach has been adopted. As a consequence, MBSE (Model-Based Systems Engineering) has become an accepted approach for designing complex systems as it allows solving systems engineering problems by expressing multiple views that can transform stakeholders' requirements into detailed specifications with the aid of models [3].

Aiming to provide simple however effective constructs for modeling a large range of systems engineering

problems, SysML allows defining high-level connections that exist between requirements, the structure and the behavior of a system. However, SysML descriptive models alone are insufficient for system behavior verifications. As a consequence, system engineers use simulation tools to conduct systems behavior analysis.

As a result, many research works have been done in order to explore how to combine SysML potential of describing systems at a high level of abstraction and simulation capabilities regarding behavior verification. Animated by the same motivation, we have performed, in an early work [4], a taxonomy of some of the most used simulation environments (i.e. Simulink, SystemC, VHDL, Modelica) and their capabilities. In this paper, we propose an automated integration approach from SysML modeling to Mathworks Simulink tool. The advantages of such integration are the compliance between SysML models that represent the studied system and the generated Simulink models, and the fact that no time (and no effort) is needed to obtain Simulink models as the system SysML model is available [5]. According to [6], there are two approaches that allow coupling UML/SysML and MATLAB/Simulink models: co-simulation and combination based on a common underlying executable language.

The proposed approach in this paper is based on co-simulation. The main objective of this approach is to generate a MATLAB/Simulink code from SysML models in order to conduct system behavior verification through simulations. To this end, a SysML4Simulink profile has been proposed to perform models transformation between SysML and Simulink. We aim implementing the proposed approach on a framework that is compliant with the MDE principles (i.e. profiles, models transformation, etc.) to ensure automated mapping between SysML and Simulink models.

This paper is organized as follows: Section 2 introduces a background of MBSE, SysML modeling and co-simulation with Simulink. In Section 3, some related works on the passage from SysML to MATLAB/Simulink will be presented before talking about our approach to combine SysML and Simulink in Section 4. An illustrative example, used to demonstrate the integration approach, is studied in section 5 before concluding and presenting our perspectives.

2. Background

2.1. Model-Based Systems Engineering

Nowadays, recent systems are more and more complex than those previously designed. Whether aircraft, new-generation cars, mobile phones, etc., the requirements for designing such systems require the combination of several disciplines (e.g. electronics, hydraulics, mechanics, etc.). Systems engineering is a multidisciplinary scientific approach that aims developing solutions in response to the requirements of different stakeholders [1]. It is a general methodological approach that encompasses all the activities necessary to design, evolve and verify a system providing an economical and efficient solution to the needs of a client while satisfying all the stakeholders [7].

In order to meet the increasing requirements of recent systems, the SE proposes a methodical approach to mastering the understanding, development and exploitation of these systems. This approach is the Model-Based Systems Engineering. The latter can be defined as a formalized modeling application for describing requirements, design, analysis, optimization, verification and validation of a system [8]. It starts from the design stage, through the development and production phases, until operations and maintenance phase.

The adoption of MBSE offers several advantages. First of all, because of its characteristics as rigor and precision, fluidity of communication between the development team and the customer, and good management complexity, this approach improves the quality/productivity ratio and minimizes potential risks during the different design phases. In addition, MBSE provides mechanisms that assign a deeper

dimension to systems engineering without increasing costs. Furthermore, data-centric specifications, which characterize model-based engineering, provide better automation and optimization that allows system engineering to focus on value-added tasks from time to time.

Due to the impact of MBSE in managing the increasing complexity of recent systems, several methodologies have been developed by industry and research teams. In [9], a taxonomy of these different methodologies, including their description and implementation is presented. As an example, we can mention IBM Telelogic Harmony-SE, INCOSE Object-Oriented Systems Engineering Method (OOSEM), Vitech Model-Based System Engineering (MBSE), etc.

In addition, MBSE has been presented as a method involving a fundamental shift from traditional document-centric approaches to computer-based models [3]. In order to support the MBSE approach, a graphical language for describing the different design phases of a system is proposed. Named SysML, this language aims overcoming UML limitations in the context of SE [10] as its inability to model parameters which change result in a different value system operation.

2.2. SysML Modeling Language

Specified by the OMG, SysML [11] is a general-purpose graphical modeling language that supports several development tasks of complex systems that may include hardware, software, data, personnel, procedures, facilities and other elements on man-made and natural systems [1]. SysML is a language that support the MBSE approach, and it is considered as a notation and not a methodology.

If UML is based on the principle of class, SysML center mainly around the notion of block. The latter allows describing both logical and physical objects of complex systems. Considered as the modular unit of structure in SysML, blocks are used to define a type of system, component, or item that flows through the system, as well as external entities, conceptual entities or other logical abstractions [1].

Systems are modeled through SysML by using diagrams for structure and behavior, as well as diagrams describing requirements and parametric relationships. Requirement diagram depicts requirements captured in SysML. A requirement specifies a capability or condition that must be (or should) be satisfied, a function that a system must perform or a performance condition a system must achieve [1]. As for Parametric diagram (PD), it is used to apply or specify systems of equations that constrain the properties of subsystems, blocks and parts.

In addition to PD, SysML structural diagrams involve Block Definition Diagram (BDD) and Internal Block Diagram (IBD). The former is used to represent blocks, their proprieties and their relationships. It allows specifying the system hierarchy and system/component classifications. The latter provides a description of the block internal structure, the type of composition and the topology of internal communications.

Through its several diagrams, SysML allows the developer to specify systems' structure and behavior as well as their requirements, and ensure software/hardware partition and performance estimation without changing the SysML modeling environment [12]. Nevertheless, it lacks of continuous time behavior's definition and executable semantics that are required for system behavior verifications. Moreover, the specification of systems' behavior is often performed separately from simulation, leading to inconsistencies in systems' development.

In order to overcome SysML semantics' lacks, the extension mechanism through the use of profiles is used by some research works as a solution [13], [14], [15]. Based on stereotypes, a profile is a mechanism used to customize UML/SysML language. As for the integration of SysML modeling and simulation, several research works [16], [17] have focused on linking SysML to one or multiple simulation environments (i.e. Modelica, SystemC, MATLAB/Simulink, etc.) in the aim of ensuring system' behavior verification.

2.3. Simulation and Co-simulation with Simulink

Considered as the most common methods that consist on using models in order to answer questions about systems whose behavior changes over time, simulation allows performing experiments and reconfigurations that may be impossible or financially expensive to realize on the real system. A simulation model of a system can be defined as any model on which an experiment (process of extracting information from a system by stimulating its inputs) can be applied to answer questions about the system [18].

The frequent use of the simulation process is related in particular to the availability of various simulation environments, that are specific to particular fields, and the development and optimization of simulation modeling methodologies. In addition, simulation is very useful and efficient for several reasons [19]. First, it allows the study and experimentation of the internal interactions of a complex system or one of its subsystems. In addition, the information obtained through the design of the simulation model can be very beneficial in proposing system improvements during the study phase. Moreover, simulation can be used as an educational tool to reinforce and verify the methodologies of analytical solutions. It should also be noted that recent systems are so complex that their internal interactions can be managed only by simulation.

In reason of their importance in testing process, several simulation tools have been created or adapted recently to deal with the increasing complexity of engineering systems projects. Considered as a proprietary development environment and programming language of The Mathworks, MATLAB (Matrix Laboratory) was designed to visualize, compute and program mathematical expressions. It allows matrix manipulations, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages (i.e. C, C++, C#, Java, etc.).

Simulink is an extension of MATLAB designed to model, simulate, and analyze dynamic systems using block diagrams. It aims designing and simulating systems before moving to hardware, and, as a consequence, reducing expensive prototypes by testing systems under conditions that are risky or time-consuming to consider. Moreover, an automatic code generation (i.e. C, HDL) can be conceivable by exploiting Simulink models in order to deploy it directly onto suitable hardware.

According to [6], there are two approaches that allow combining SysML with Simulink: Co-simulation or executable common language. The latter consists on generating C/C++ code from MATLAB/Simulink and link it to a C++ implementation of UML/SysML model. This approach has the advantage of allowing faster simulation speed. As for the co-simulation approach, SysML tool and Simulink interact with each other through a coupling tool.

As SysML is more powerful with regard to requirements modeling and the overall system design, MATLAB/Simulink has its strengths in the simulation area. Therefore, many research works have focused on joining both SysML modeling, and behavior verification of systems with Simulink (or other simulation environments) to assist system designers and control engineers in order to accelerate the design and testing processes.

3. Related Works

In order to benefit from capabilities and strengths offered by combining both SysML modeling and simulation with MATLAB/Simulink, various research works have explored their integration. In this context, the authors of [12] and [20] have explored the integration between UML/SysML modeling and embedded systems simulation and synthesis by defining an approach based on code generation. This approach consists on SysML extensions for SystemC, C/C++ and MATLAB/Simulink co-modeling and co-simulation. Furthermore, after successful simulation, a VHDL synthesis can be envisaged on synthesizable SystemC code.

In order to allow system engineers to verify specification of a control system whose behavior is described by mixing both continuous-time and event-driven models, the authors of [21] proposed an extension of

SysML which enables describing continuous-time behavior. Furthermore, they developed its execution tool by exploiting co-simulation of SysML and MATLAB/Simulink.

In [5], the authors have described an automatic model transformation approach aimed at obtaining Simulink models conform to SysML source models. The proposed workflow is based on an M2T transformation that translates a source SysML model into a MATLAB model generation script.

As for the authors of [3], they explored how to create, sync and generate the SysML design model and the Simulink/Simscape compatible models in order to carry out the simulation automatically. Therefore, the Simulink-compatible model was first defined to specify the system continuous-time behavior before extending the SysML semantics through a set of stereotypes. Secondly, to enable an automatic simulation, a SysML-based simulation model has been performed for use in the Simulink environment.

The authors of [22] described a tool to integrate SysML with Simulink and to evaluate parametric diagrams. Their objective is based on combining a SysML modeling tool (IBM Rational Rhapsody) with a proprietary simulation tool (Mathworks Simulink) and a Computer Algebra System (CAS) to validate system specification.

Compliant with MDE principles (i.e. profile mechanism, model transformation, etc.) and based on co-simulation between SysML and Simulink models, the integration approach proposed in this paper is oriented in the same context as the latter works aiming to combine the potential of both SysML modeling and MATLAB/Simulink simulation.

4. Passage From Sysml To Simulink

The integration approach, aiming to link SysML to Simulink, is handled by MDE principles. MDE is considered as a generative engineering form in which models occupy a prominent place among systems development artefacts since they allow generating all or part of these systems [23]. It promotes to provide a large number of models to describe separately different concerns of a system. Thus, the concept of model is considered as MDE central one. However, in order to be characterized as productive, a model must be handled by a machine. Therefore, the language with which this model is described must be clearly defined. As a consequence, the definition of a modeling language took the form of a model, called metamodel [24].

Model transformation constitutes a fundamental process in the MDE approach as it aims providing an operational aspect to models (for code, documentation and test generation, validation, verification, execution, etc.). It can be defined as an operation that automatically creates a set of target models from a set of source models [25]. The impact and efficiency of model transformation is amplified by the need of defining modeling languages dedicated to a particular domain. As a fact, the definition of Domain Specific Modeling Languages (DSML) is promoted by MDE due to well-know advantages [26] offered by domain-specific languages as the fact that dedicated languages allow validation at the domain level.

In order to ensure the passage from SysML to Simulink, the integration approach proposed in this paper is based on a MATLAB/Simulink code generator from SysML models. This code generator consists on several model-oriented tools as:

EMF[1] **:** The Eclipse Modeling Framework [27] is a project that allows modeling and code generation facility for building tools and other applications based on a structured data model. Using a model specification specified in XMI, EMF offers possibilities through tools and runtime support in order to produce a set of Java classes for the model, in addition to a set of adapter classes that allow viewing and command-based editing of the model, and a basic editor.

Papyrus[2] **:** This graphical editing tool has been adopted to provide a complete support to SysML

[1] https://eclipse.org/modeling/emf/

[2] https://eclipse.org/papyrus/

modeling and, thus, to ensure model-based system engineering. Papyrus is a open-source UML2.0 [28] tool based on Eclipse that can be used as a standalone tool or as as an Eclipse plugin, providing support for Domain Specific Languages and SysML.

Acceleo[3] **:** Regarding the transformation from SysML models to MATLAB code, it consists on Acceleo language which is an Eclipse implementation of the OMG MOF2Text Transformation Language (MTL) [29]. Acceleo is considered as a language code generator that allows generating structured file from an EMF model. This generated file consists on a text that can be a programming language or other formalism. In this context, Acceleo requires the definition of both EMF metamodel and a model conforming to metamodel that will result into text [30].

Fig. 1 depicts the main elements that allow conducting MATLAB code generation based on Acceleo tool. Acceleo templates (.mtl) are used to define transformation rules performed on SysML source models. These latter are conformant to a "SysML4Simulink" profile extended with new and adequate stereotypes in order to ensure a better mapping between SysML and Simulink. In order to detail the functioning of the Acceleo code generator, we propose in this section a presentation of "SysML4Simulink" profile, enriched with stereotypes for optimizing the passage from SysML to Simulink, in addition to model transformation rules specified in the context of the "SysML2Simulink" transformation process.

4.1. SysML4Simulink Profile

Through its diagrams, SysML allows describing requirements, structure and behavior to provide a complete description of a system, its components and its environment. Blocks are considered as the most basic structural elements of SysML as they allow representing systems, subsystems, parts, etc. SysML diagrams that use mainly blocks are Block Definition Diagram (BDD), Internal Block Diagram (IBD) and Parametric Diagram (PD).

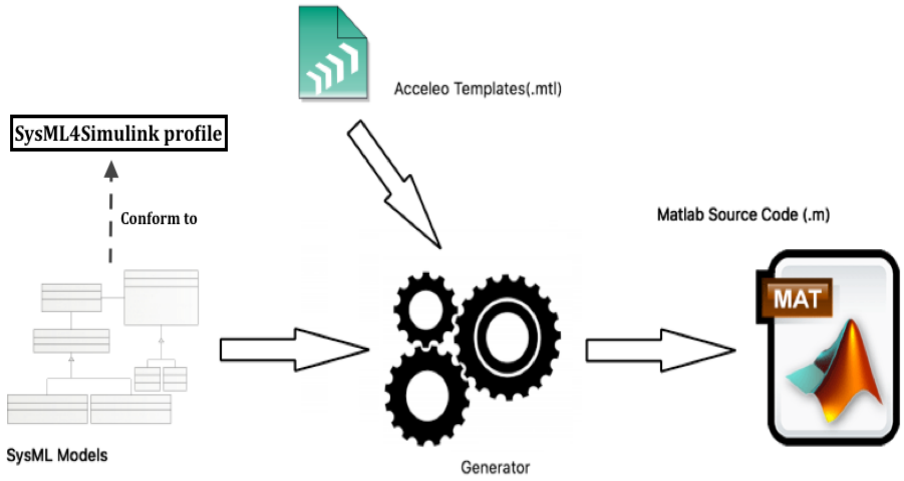

Fig. 1. MATLAB code generation approach.

SysML structural diagrams and their constructs are the most corresponding entities to describe Simulink language constructs. This is due to the facts that SysML structural diagrams allow:

Describing systems and their elements, in addition to connectors between blocks and flows that flow through them. As for Simulink, this description concern components and subsystems, as well as links and directional and bidirectional flows between them;

Specifying blocks' behavior through the use of constraint blocks. This aspect is corresponding to the

[3] http://www.eclipse.org/acceleo/

specification of components behavior through equations in Simulink;

Support reusing of the same element by several diagrams.

As a consequence, due to the fact that SysML structural diagrams are the closest to Simulink models, SysML4Simulink profile modeling is performed through three diagrams: BDD, IBD and PD.

Nevertheless, some semantic differences between SysML and Simulink remain and affect their integration. In this context, SysML value properties does not allow indicating whether their values are constant or variable, as it is the case in Simulink. Furthermore, if blocks are used in SysML structural diagrams to describe systems as well as subsystems and parts, Simulink uses several constructs (subsystem, component, etc.) to specify the model hierarchy.

These semantic differences can be overcoming through the extension of SysML4Simulink profile with new semantics. This extension is based on the use of UML's profiling facility, especially stereotypes. The latter are an extensibility mechanism of UML/SysML which allows users to define modeling elements derived from existing UML classifiers, such as classes and associations, and to customize these towards individual application domains [28]. In this context, SysML4Simulink profile has been enriched with three stereotypes as shown in Fig. 2, in order to ensure an effective mapping between SysML and Simulink constructs:

"SimulinkConstant" stereotype: As SysML does not allow the distinction between variables and constants, this stereotype that inherits SysML property is used to indicate that a value remains constant at each simulation run.

"SimulinkBlock" stereotype: Specified as a special kind of SysML Block, this stereotype is particularly useful for describing interactions between the components in terms of simulation tools, but can also be used to describe the components themselves, if necessary.

"SimulinkContentBlock" stereotype: Defined as a special kind of SysML Block, its role is to encompass the whole system.

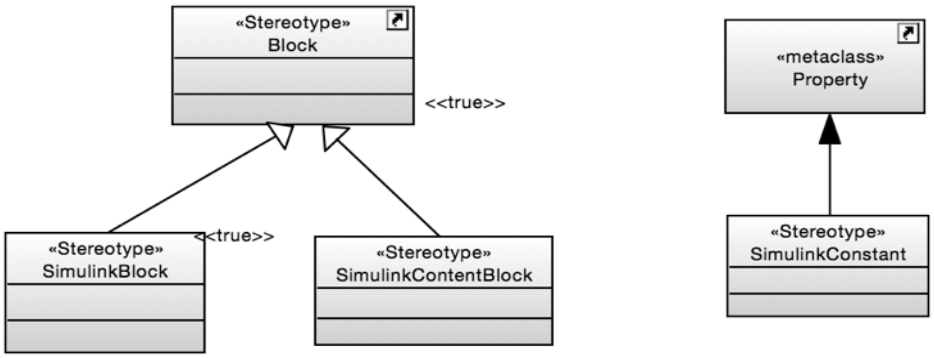

Fig. 2. SysML4Simulink stereotypes.

The SysML4Simulink profile allows SysML modeling of systems in order to ensure their behavior' verification through the proposed code generator. As a fact, SysML models specified via this profile are considered as inputs of the code generator that will be transformed to Simulink models conforming to transformation mapping defined through Acceleo templates.

4.2. SysML2Simulink Transformation

In general, the definition of a model transformation requires manipulation of elements defined in both source and target metamodels. Therefore, the implementation of a model transformation requires a good

knowledge of metamodels, and the link between the elements defined in the metamodel and concrete syntax when it exists. Since there is no available metamodel of the Simulink language, we are constrained to perform an M2T (Model-To-Text) transformation through the use of Acceleo tool.

The objective is to translate SysML models into an executable MATLAB code. The latter is used to generate Simulink models through MATLAB application. It is to note that Simulink models have the same expressiveness of the source SysML models.

Table 1. SysML4Simulink Profile and Simulink Constructs Mapping

SysML4Simulink profile	Simulink	Note
SimulinkContentBlock	Block (System)	SimulinkContentBlock turns into the complete Simulink system
SimulinkBlock	Subsystem Block	Each part of a SysML SimulinkContentBlock is modeled with a Simulink Subsystem block
FlowPort	Input/Output block	SysML FlowPorts are modeled with Simulink Input and Output Blocks
Constraint Block	Dynamic equation	SysML constraints are modeled in Simulink using dynamic equations
Connectors	Lines	Used to connect ports, SysML connectors are modeled in Simulink by lines
FlowSpecification	Bus Object	Typing SysML FlowPort, FlowSpecification are modeled by Simulink Bus Object

The proposed integration process between SysML and Simulink allows exploiting the advantages offered by both system engineering and simulation. The automatic SysML to Simulink transformation permits obtaining Simulink models that are conformed to source SysML ones, in addition to the fact that since SysML models are available, the correspondent Simulink models are generated instantaneously and without effort.

The first step in our transformation process is to specify transformation rules from SysML to Simulink by defining a mapping between SysML and Simulink constructs. Defined mapping for a specific domain can be used in the development of several applications in the same domain. Table I summarizes the mapping applied for the transformation from SysML to Simulink.

The mapping applied is then used to create Acceleo templates. The latter are texts containing spaces in which source model information will be placed. Fig. 3 illustrates a part of Acceleo templates. The template on the left represents the generation of the main Simulink model, through the transformation of the stereotype SimulinkContentBlock to Simulink Block (generateBlock). As for the template on the right of Fig. 3, it specifies the generation of Simulink subsystem block, obtained from the stereotype SimulinkBlock. As a fact, the template details the creation of graphical aspects of the block (i.e. name, size, etc.), and the definition of its ports, properties and equations.

As illustrated inFig. 1, MATLAB code generator is based on both SysML4Simulink profile and Acceleo

templates. Once the SysML model that conforms to SysML4Simulink profile is achieved, it is sent as input to the code generator. The latter transforms each element of SysML source model into its equivalent as it has been defined in Acceleo templates. At the end, executable MATLAB code in the form of script files (.m) is obtained to be used to generate Simulink model through MATLAB application. As it will be shown in the next section, the Simulink model obtained is conform to SysML source model.

```
[comment encoding = UTF-8 /]
[**
 * The documentation of the module generate.
 */]
[module generate('http://www.eclipse.org/papyrus/0.7.0/SysML','
[import operations/]

[**
 * The documentation of the template generateElement.
 * @param anAllocate
 */]
[template public generateElement(b : Model)]
[comment @main/]
  [for (p: Element | self.ownedElement) ]
    [if (p.oclIsTypeOf(Package))]
      [for (e : Class | p.ownedElement)]
          [generateBlock(e,b)/]
          [generateSimulinkContentBlock(e,b) /]
      [/for]
    [/if]
  [/for]
[/template]
```

```
[template public generateBlock(block : Class, model : Model)
     ? (block.hasStereotype('simulinkBlock')) ]

   [if (block.oclIsTypeOf(Class))]
    [file (block.name+'_generator.m', false, 'UTF-8')]

     function [block.name/]
x = 30; y = 30; w = 30; h = 30;
offset = 60;
fname = '[block.name/]';

if exist(fname,'file') == 4
    if bdIsLoaded(fname)
        close_system(fname,0)
    end
    delete(fname,'.mdl');
end

new_system(fname);
       [for (port : Port | block.ownedPort)]
          [generatePort(port)/]
       [/for]
       [for ( prop :Property | block.ownedAttribute)]
          [generateConstansProp(prop)/]
       [/for]
   [ generateEquations (model,block.name)/]
save_system(fname);
    [/file]
 [/if]
[/template]
```

Fig. 3. Example of an Acceleo template.

In order to illustrate the effectiveness and the use of the proposed integration approach, an illustrative example is shown in the next section. It concerns the study of an electrical circuit. The latter is modeled using SysML structural diagrams. These diagrams are sent to the proposed MATLAB code generator in order to obtain correspondent Simulink models, used to perform simulations for conducting system behavior' verification.

5. An Illustrative Example

5.1. System Description

Fig. *4* illustrates the studied electrical circuit. It consists on a source of electricity (i.e. battery), a chopper and a direct current (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 obtaining a slot-shaped voltage and an adjustable average value from a fixed voltage, and this, 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 α can be defined as :

$$\alpha = \frac{t_F}{T} \quad \text{with} : 0 \leq \alpha \leq 1$$

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

- If $0 \leq t < t_F$, the chopper switch is closed: Vs = E;

- If $t_F \leq t < T$, the chopper switch is opened: Vs = 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 electromechanical converter that allows bidirectional conversion of energy between an electrical installation traversed by a direct current and a mechanical device. As a 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. 5 depicts some of the requirements for the DC motor operation. In the following study, we focus on the verification of the rotor rotation' requirement, stipulating that the rotor rotational speed must increase continuously according to DC output voltage of the chopper.

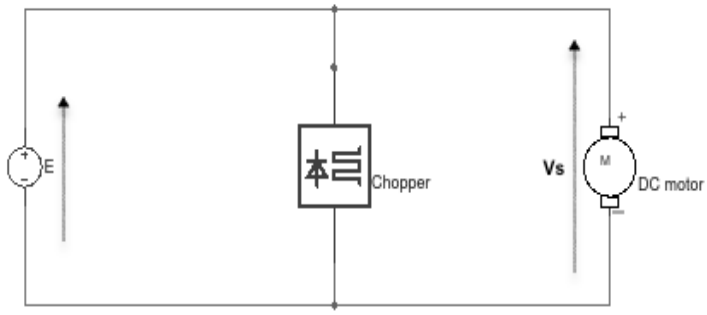

Fig. 4 Schematic of the studied electrical circuit

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 has two magnetic circuits, called stator (fixed part) and rotor (moving part). In the case of the DC motor, the stator, also called inductor, creates a magnetic field; while 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 this relation:

$$E_{cef} = K.n \quad (2)$$

K is a constructor datum 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:

$$V_s = E_{cef} + R.I \cong E_{cef} \quad (3)$$

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:

$$n = \frac{Vs_{moy}}{K} \quad (4)$$

$$n = \frac{\alpha.E}{K} \quad (5)$$

5.2. SysML Modeling of the Studied System

In order to apply the proposed integration approach, the studied system is modeled using SysML structural diagrams. Thus, Fig. 6 depicts the BDD that models the electrical circuit. The latter is represented by the stereotype "SimulinkContentBlock". As for the three components of the circuit, that are the battery, the chopper and the DC motor, they are modeled using the stereotype "SimulinkBlock".

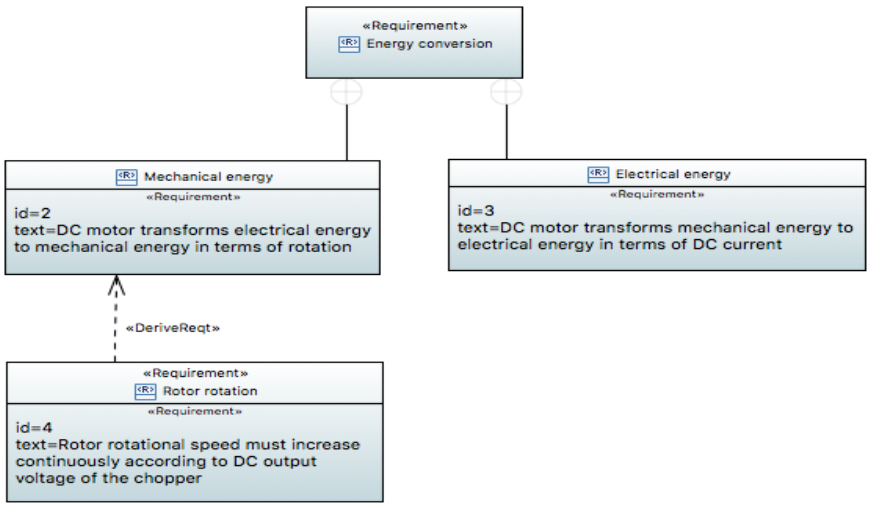

Fig. 5 Requirement diagram of the DC motor.

The BDD allows representing flow ports that permit exchanging properties between the subcomponents of the electrical circuit. As a fact, the chopper block receives the battery source voltage E in order to calculate the average voltage between the chopper terminals Vsmoy. The latter is sent and received by the DC motor block in order to obtain the value of rotor rotational speed n. Equations (1) and (5) are both specified through constraint blocks. It is to note that the cyclic report α is modeled as a variable in the SysML BDD, and defined through an OCL constraint in order to specify its value variation between 0 and 1.

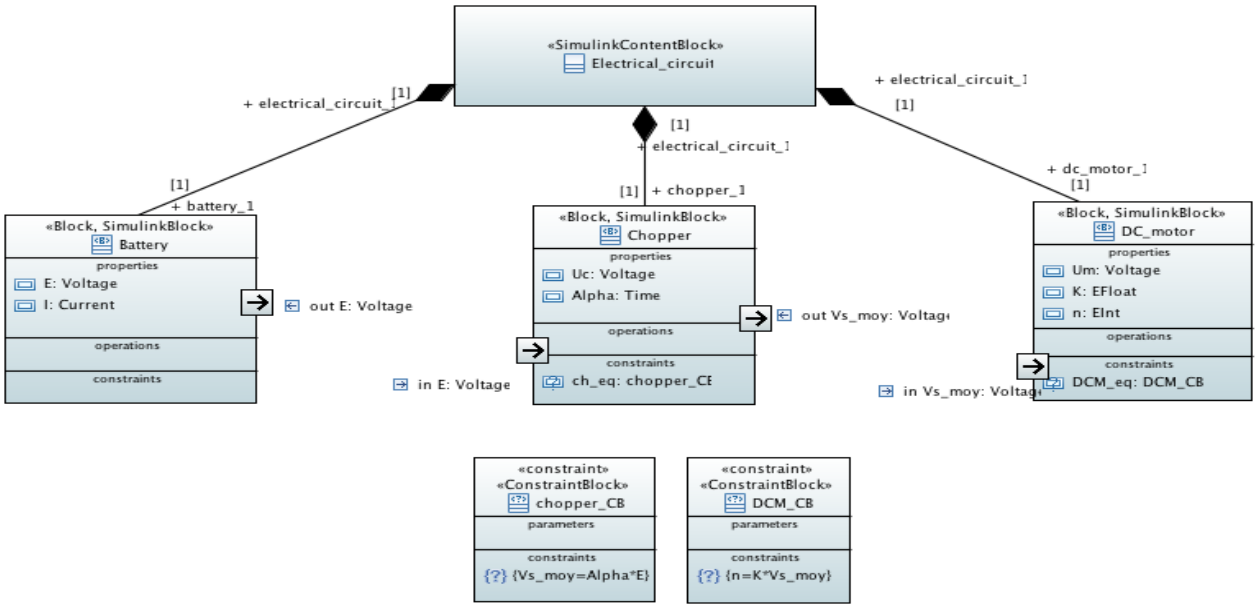

Fig. 6. The BDD of the electrical circuit.

Fig. 7 illustrates the IBD of the studied system. The IBD allows representing the flows exchanged between the electrical circuit parts (i.e. the voltage), by the specifying a set of architectural connectors between the identified blocks in the BDD. Therefore, the subcomponents of the electrical circuit are represented as IBD parts, that are linked through SysML connectors. The latter are connections established between the parts ports in order to allow exchanging information, as depicted in Fig. 7.

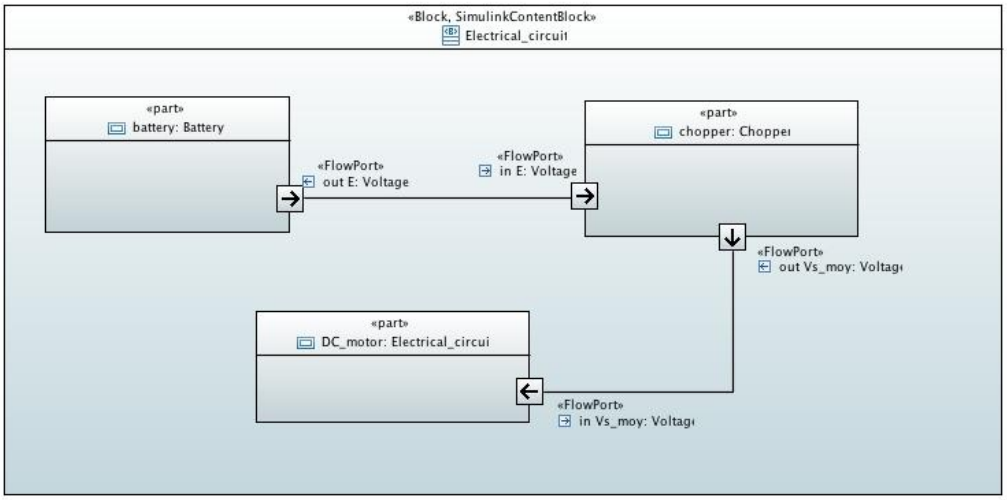

Fig. 7. An IBD of the electrical circuit.

As depicted in Fig. 8, the PD of the electrical circuit permits specifying the behavior of both chopper and DC motor through the specification of equations (1) and (5), and the link between them. As a fact, the constraint blocks, defined early in the BDD, are represented as constraint properties in the PD, in addition to equations that specify the behavior of both chopper and DC motor. Furthermore, the PD permits modeling the connection between these constraint properties, through which information is exchanged (Vs_{moy}).

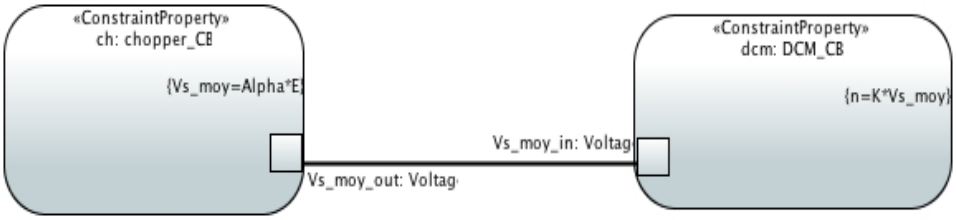

Fig. 8. Parametric diagram of the electrical circuit.

5.3. Simulation of the Generated Simulink Model

In order to apply the transformation process of our proposed approach, SysML model of the studied system is sent as an input of the MATLAB code generator. The output of this latter is script files (.m) that contain executable MATLAB code. Once imported in a MATLAB application, the generated script files are executed to obtain the equivalent Simulink model. Therefore, Fig. 9 depicts generated Simulink model automatically obtained from SysML source model. As a fact, the main variables and operators described through SysML constraint properties are modeled via the Simulink model. Furthermore, it is to note that the cyclic report α is modeled as a Simulink ramp, in order to specify its value variation during the simulation.

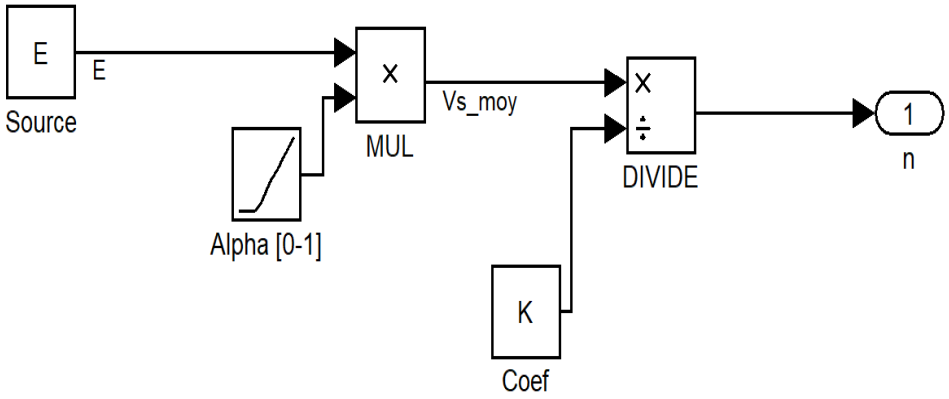

Fig. 9. Generated Simulink model of the studied electrical circuit.

As it is specified earlier, the 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, in order to verify the rotor rotation requirement of Fig. 5 As a consequence, we have performed a simulation using the obtained Simulink model in order to observe the variation of DC motor's rotation speed (n) according to variable DC output voltage of the chopper (Vs_{moy}). Since the source voltage E is equal to 12V and the constant of speed K is estimated to 0.0458, the rotor rotational speed varies as illustrated in Fig. 10. As a fact, we can notice that this parameter increases continuously according to the DC output voltage of the chopper. Furthermore, this simulation allows conducting experiences that permit to observe and study the DC motor rotational speed according to various input parameters (i.e. E, K, Vs_{moy}, etc.).

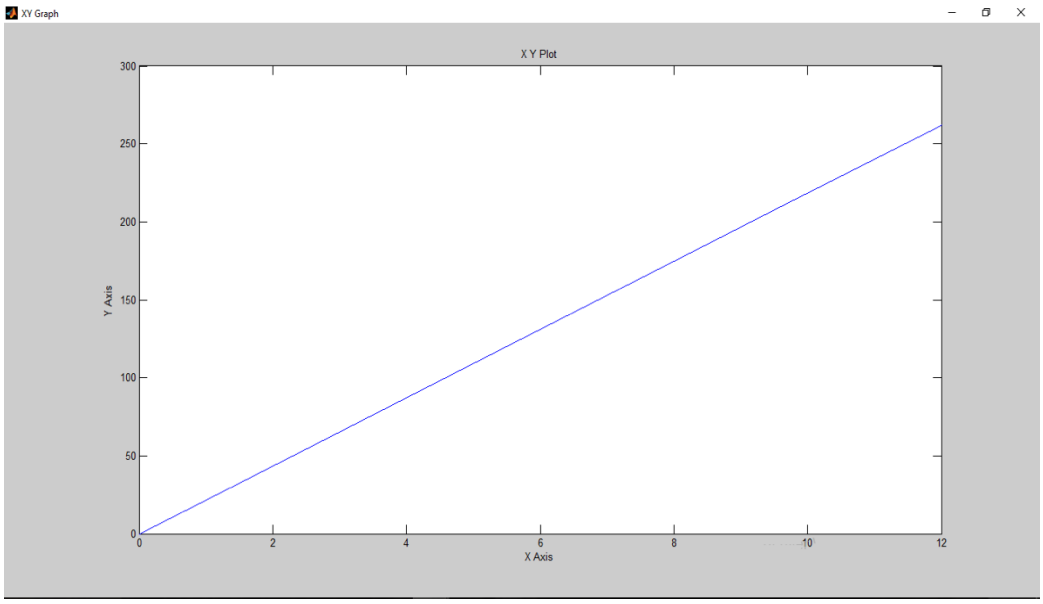

Fig. 10. Simulation for DC motor rotational speed monitoring.

6. Discussion and Conclusion

In the context of System Engineering, SysML has intrinsic capabilities in terms of recent systems' description. However, its descriptive models remain insufficient to verify the behavior of these systems.

Moreover, SysML modeling does not allow describing behaviors with continuous time, but rather with discrete events. These aspects are covered by simulation process, through the use of various environments developed by manufacturers and research teams, as MATLAB/Simulink. In order to benefit from the advantages of both SysML modeling and Simulation of recent systems, several research works have focused on combining them through different integration approaches.

In this paper, we have presented our integration approach based on model transformation profile to generate Simulink models from SysML source models.

The proposed integration approach is based on MDE principles. It consists on SysML modeling of a system by means of SysML4Simulink profile that was extended with special stereotypes. The passage from SysML to Simulink is guaranteed by a transformation process whose rules are defined through the establishment of mapping between SysML and Simulink constructs. Through SysML2Simulink transformation, an executable MATLAB code is generated from SysML source diagrams and used to perform Simulink models that will allow conducting simulations in order to verify system behavior.

Even though the integration approach, proposed in this paper, ensures an automatic passage from SysML models to Simulink ones, allowing, as a consequence, system behavior' verification, however, the approach can be optimized through the study of the following axis. The first one concerns the fact that this approach is semi formal, even if it is handled by MDE principles. This is due to the fact that the passage between SysML4Simulink profile and Simulink is based on an M2T transformation process. The latter is used since there is no metamodel for the Simulink language that can permit establishing an M2M transformation process between SysML4Simulink profile and Simulink metamodel.

The second axis to improve in the proposed approach concerns the unidirectionality of the passage between SysML and Simulink. As a fact, our integration approach remains unidirectional since we have not yet define the passage from Simulink models to SysML models. We are working to establish this passage to benefit fully of the potentials and strengths of combining SysML and Simulink.

This paper represents an important step in our research works. Indeed, we are studying how to bridge the gap between SysML and several simulation environments including MATLAB/Simulink through a model-driven integration platform, based on MDE principles, that ensures bidirectional transformation between these environments and SysML. To this end, we define an intermediate language in order to link SysML modeling and simulation environments. Named SimulML (Simulation Modeling Language) [31], this language is defined in the basis of common constructs, semantics and modeling methodologies of the most used simulation environments. As a fact, both the language and the integration process will be treated by future works.

References

[1] Friedenthal, S., Moore, A., & Steiner, R. (2012). A practical guide to SysML: the systems modeling language. Waltham, MA: Morgan Kaufmann, 2012.

[2] Weilkiens, T. (2006). Systems engineering with SysML/UML: Modeling, analysis, design. Heidelberg: Morgan Kaufmann, 2006.

[3] Rahman, M. A. A., & Mizukaw, M. (2013). Model-based development and simulation for robotic systes with SysML, simulink and simscape profiles, *Int. J. Adv. Robot. Syst.*, *1*.

[4] Chabibi, B., Anwar, A., & Nassar, M. (2015). Towards an alignment of SysML and simulation tools. presented at the 12th ACS/IEEE International Conference on Computer Systems and Applications AICCSA, Marrakech, Morocco.

[5] Sindico, A., Natale, M. D., & Panci, G. (2011). Integrating SysML with simulink using open-source model transformations.

[6] Vanderperren, Y., & Dehaene, W. (2006). From UML/SysML to matlab/simulink: Current state and future perspectives. *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings*.
[7] Roques, P. (2013). Modélisation de systèmes complexes avec SysML.
[8] Kaslow, D., Graphics, A., Soremekun, G., Kim, H., & Spangelo, S. (2014). Integrated model-based systems engineering (MBSE) applied to the simulation of a cubesat mission. *Proceedings of the Aerospace Conference*.
[9] Estefan, J. A. (2007). Survey of model-based systems engineering (MBSE) methodologies. *Incose MBSE Focus Group*.
[10] Bézivin, J. (2003). On the unification power of models.
[11] Object Management Group. (2012). OMG systems modeling language.
[12] Vanderperren, Y., Mueller, W., He, D., Mischkalla, F., & Dehaene, W. (2012). Extending UML for electronic systems design: A code generation perspective. *Design Technology for Heterogeneous Embedded Systems*.
[13] Johnson, T., Kerzhner, A., Paredis, C. J. J., & Burkhart, R. (2011). Integrating models and simulations of continuous dynamics into SysML *J. Comput. Inf. Sci. Eng*.
[14] Shah, A. A., Kerzhner, A. A., Schaefer, D., & Paredis, C. J. (2010). Multi-view modeling to support embedded systems engineering in SysML. *Graph Transformations and Model-driven Engineering*.
[15] Fritzson, P., Rouquette, N. F., & Schamai, W. (2010). An overview of the sysml-modelica transformation specification.
[16] Vasaiely, P. (2009). Interactive simulation of SysML models using modelica. Hamburg university of applied sciences, Faculty of engineering and Computer Science.
[17] Cafe, D. C., Santos, F. V. D., Hardebolle, C., Jacquet, C., & Boulanger, F. (2013). Multi-paradigm semantics for simulating SysML models using SystemC-AMS. *Specification and Design Languages*.
[18] Fritzson, P. A. (2011). Introduction to modeling and simulation of technical and physical systems with Modelica.
[19] Jerry, B. (2005). Discrete-event system simulation.
[20] Mischkalla, F., He, D., & Mueller, W. (2010). A UML profile for SysML-based co-modeling for embedded systems simulation and synthesis. *Proceedings of the Workshop on Model Based Engineering for Embedded System Design (MBED)*.
[21] Kawahara, R. *et al.* (2009). Verification of embedded system's specification using collaborative simulation of SysML and simulink models. *Proceedings of the International Conference on Model-Based Systems Engineering*.
[22] Sakairi, T., Palachi, E., Cohen, C., Hatsutori, Y., Shimizu, J., & Miyashita, H. (2013). Model based control system design using sysml, simulink, and computer algebra system. *J. Control Sci. Eng*.
[23] Bézivin, J., & Blanc, X. (2002). Promesses et interrogations de l'approche MDA. *J. Développeur Réf*.
[24] Combemale, B. (2008). Approche de métamodélisation pour la simulation et la vérification de modèle – Application à l'ingénierie des procédés. *Institut National Polytechnique de Toulouse-INPT*.
[25] Jouault, F. (2006). Contribution à l'étude des langages de transformation de modèles. *Informatique, Université de Nantes*, UFR Sciences & Techniques, 2006.
[26] Jézéquel, J.-M. Combemale, B., & Vojtisek, D. (2012). Ingénierie Dirigée par les Modèles : des concepts a la pratique.
[27] Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2008). EMF: Eclipse modeling framework.
[28] Object Management Group. (2007). Object management group.
[29] Specification, O. M. G. (2007). A uml profile for marte, beta 1. *Object Manag. Group Pct07-08-04*.

[30] Abdulhameed, A., Hammad, A., Mountassir, H., & Tatibouët, B. (2014). An approach combining simulation and verification for SysML using systemc and uppaal. presented at the CAL 2014, 8ème conférence francophone sur les architectures logicielles.

[31] Chabibi, B., Anwar, A., & Nassar, M. (2016). Metamodeling approach for creating an abstract representation of simulation tools concepts. presented at the 13th ACS/IEEE International Conference on Computer Systems and Applications AICCSA.

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 PhD 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 is currently an associate professor in computer science at the university of Mohammed-V in Rabat and as 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-formals methods in software and system's development.

Mahmoud Nassar is a full professor at National Higher School for Computer Science and Systems Analysis (ENSIAS), Rabat, Morocco. He is a head of IMS (IT architecture and Model Driven Systems development) Team / ADMIR Laboratory of Rabat IT Center. He received his PhD in Computer Science from the INPT Institute of Toulouse, France. His research interests are 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.