Model Integration Approach from SysML to MATLAB/Simulink

Bassim CHABIBI[1], Adil ANWAR[2], Mahmoud NASSAR[3]
[1, 3] IT architecture and Model driven Systems development (IMS)
Labo. ADMIR, Rabat IT Center
ENSIAS, Rabat Mohamed V University in RABAT
Morocco
bassim.chabibi@um5s.net.ma
nassar.ensias@gmail.com

[2] SIWEB, E3S
EMI, Rabat Mohamed V University in RABAT
Morocco
anwar@emi.ac.ma

Journal of Digital Information Management

ABSTRACT: *In system engineering process, descriptive system models seem to be insufficient in order to perform a system verification which fulfils various stakeholders' requirements. This aspect is well handled by simulation process through the use of several simulation techniques or algorithms. As a consequence, design process efficiency is considerably reduced by the fact that both system modeling and simulation tools are often used separately. This study introduces an integration process to unify the potential provided by systems modeling languages and simulation environments, through the definition of a Domain Specific Language, namely Simulation Modeling Language, that is built on the basis ofa deep study of common constructs, semantics and modeling methodologies of several simulation environments, in addition to the specification of a model transformation between this language and a simulation environment (MATLAB/SImulink) in order to illustrate both its importance et its efficiency in our integration approach. Through the specification of its syntaxes and semantics, the defined intermediate modeling language allows modeling systems by using common constructs and modeling methodologies of simulation process in order to ensure their modeling with simulation environments and, thus, conduct experiences and system verifications. The definition of this language constitutes the basis of our integration approach aiming to bridge the gap between system modeling and simulation aspects in order to benefit from the strengths and potentials of both approaches. The integration approach consists on the specification of a bidirectional transformation, based on the concepts of Model-Driven Engineering, to perform in future works.*

Subject Categories and Descriptors
I.6 [Simulation and Modeling] I.6.5 [Model Development]; Modeling methodologies **; F.3.2 [Semantics of Programming Languages]**

General Terms: Modeling language, Simulation, System Engineering, Model Driven Engineering

Keywords: System Engineering, MDE, SysML, Simulation, DSL, Model Transformation

Received: 21 March 2018, Revised 8 May 2018, Accepted 27 May 2018

DOI: 10.6025/jdim/2018/16/6/289-307

1. Introduction

A complex system can be defined as a set of interacting elements. It can also be seen as a set that interacts with the outside environment. Systems Engineering (SE) is a multidisciplinary scientific approach designed to develop solutions in response to the demands of different stakeholders (Friedenthal, Moore, & Steiner, 2012). It focuses on defining customer needs and system requirements, detected early in the life cycle, through requirements documenting, design synthesis and system validation. By integrating several disciplines, SE describes a structured development process, going from system specification until its development. It takes into account technical and economic aspects in order to ensure the development of a system that meets users' needs.

Due to the increasin g complexity of recent systems, it is more and more difficult to verify the system requirements, expressed by stakeholders, by using only descriptive models. In order to help engineers to overcome this constraint, a model-based approach was adopted. As a consequence, Model-Based Systems Engineering (MBSE) has become an unavoidable approach for designing complex systems as it solves the problems of system engineering through multi-views expression that transforms stakeholders requirements into detailed specifications using models(Abdul Rahman & Mizukaw, 2013).

SysML (System Modeling Language) (Object Management Group, 2012) supports the practice of MBSE used to develop solutions in response to complex and technologically difficult problems(Friedenthal et al., 2012). It allows to represent systems including a combination of hardware, software, data, individuals, installations and natural objects. Moreover, SysML allows the development of system solutions in response to complex and technological constraints. Considered as a language for graphical description of complex systems,SysML supports their analysis, specification, design, verification and validation. Therefore, it should be recalled that SysML is a notation and not a methodology.

Nevertheless, although SysML descriptive models are suitable for the definition of the high-level links that exist between requirements, structure and behavior of a system, they do not allow the verification of system behavior. In order to remedy this lack, designers use a variety of simulation tools that allow them to address these issues without being obliged to use the real system, as such experiences can be both expensive and dangerous.

Models simulation is one of the most common techniques that use models to answer questions about systems whose behavior changes over time. 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 (Fritzson, 2011). Depending on how the model is represented, it can be categorized as mental, verbal, physical or mathematical. The latter is a description of a system whose variables are linked by a mathematical relationship.

Due to the increasing complexity of systems to be designed and the importance of simulation in the development process, various simulation environments have been developed by industry and research teams. Among these environments, we can name Modelica, VHDL, MATLAB/Simulink, Arena, AutoMod, etc. As a consequence, some research works (Arunachalam, Zalila-Wenkstern, & Steiner, 2008), (Ereeþ, Kuruoðlu, & Moralý, 2013) focused on the defining approaches that allow evaluating simulation environments in function of several criteria (e.g. code reusability, animation, documentation, etc.).

Considered as a proprietary development environment and programming language of the Mathworks, MATLAB (Matrix Laboratory) was designed to visualize, compute and program mathematical expressions. Simulink is an extension of MATLAB designed to model, simulate, and analyze dynamic systems using block diagrams.

Since system behavior is upgradeable over time, it can be studied by the definition of a simulation model. The latter is a representation of a system in order to study it, and understand the relationships between its components or to predict its behavior under a new constraint. In addition, simulation process has several other advantages (Jerry, 2005), such as testing new hardware designs without committing resources for their acquisition and exploration of new policies, operating procedures, information flows, etc., without disrupting ongoing operations of the real system.

If SysML is performant in describing systems with a high level of abstraction, simulation has intrinsic capabilities for verifying systems behavior. As a fact, several research teams have focused on combining modeling with SysML and simulation. The goal is to bridge the gap between SysML modeling and one or more simulation environments. In a previous work(Chabibi, Anwar, & Nassar, 2015), we have developed a taxonomy of simulation environments (like MATLAB/Simulinkâ, SystemCâ, VHDLâ, Modelicaâ) that have been combined with modeling SysML through a literature review. Similarly, we proposed, in *(Chabibi, Douche, Anwar, & Nassar, 2016)*, an integration approach based on co-simulation of SysML modeling and MATLAB/Simulink simulation environment.

2. Background

2.1 Model-Driven Engineering

MDE is considered as a form of generative development approach where models occupy a prominent place among systems development artifacts since they allow generating all or part of these systems. To this end, their accuracy and richness are indispensable to ensure their

interpretation and transformation by machines.

Moreover, MDE promotes providing a large number of models to describe separately each of the concerns of users, designers, architects, etc. The notion of model is considered to be the central concept of MDE for which no universal definition has been proclaimed.

In order to create a productive model, the latter must be able to be manipulated by a machine. To this end, the language with which this model is described must be clearly defined. Therefore, the definition of a modeling language has taken the form of a model, called a metamodel.

A metamodel can be defined as a specification model for which the studied system is considered as models in a certain modeling language (Seidewitz, Sept.-Oct.). In other words, a metamodel is a model of a language that describes its concepts and properties, its textual and/or graphical syntax and its semantics. A language is characterized by both its syntax and its semantics (Figure 2, extracted from (Combemale, 2008)). In the context of the MDE, the syntax of a language is defined through the specification of two parts: abstract syntax (AS) and concrete syntax (CS). AS is the internal representation of the model, manipulated by the computer. Therefore, it is defined at first place and considered as the essential basis for the definition of concrete syntax. As for CS, it consists in defining a set of decorations (textual or graphic) as well as a link between the concepts of AS, and their correspondence in decorations in CS. Concerning the semantics of a language, it designates the link between a signifier (in this case, a model) and a signified (e.g. mathematical object) in order to give a meaning to each construct of the language(Jézéquel, Combemale, & Vojtisek, 2012).

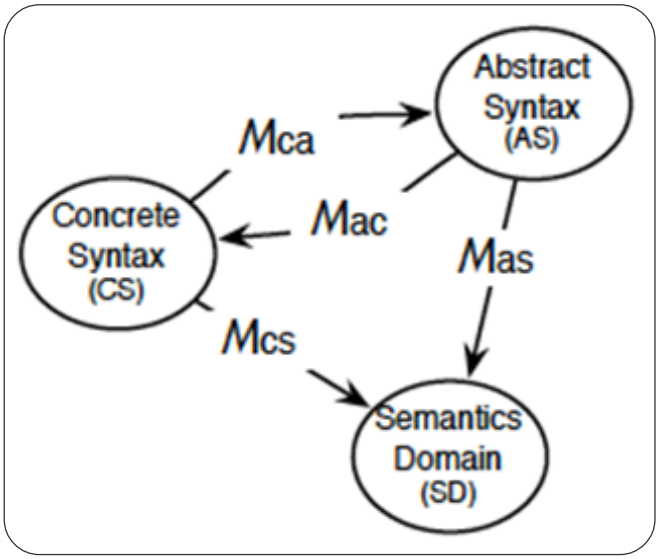


Figure 1. Definition of a language

Furthermore, MDE promotes the definition of Domain Specific Languages (DSLs). The latter are “small” specific languages that are suitable to particular corporations or needs (AS MDA, 2005). Due to the promotion of the use of DSL concept, the number of small modeling languages for specific domains are constantly growing. Therefore, the first problem facing MDE has been to regulate the definition of these DSLs. In this context, the definition of a language through metamodeling has proved to be effective and practical. The second problem is to attribute an operational aspect to the models generated via DSLs in order to optimize their manipulation. Thus, the notion of models transformation has imposed itself as a solution to this problem.

In order to provide an operational aspect to models (e.g. for code generation, documentation and testing, validation, verification, execution, etc.) model transformation is placed at the heart of MDE approach. It can be defined as an operation that automatically creates a set of target models from a set of source models(Jouault, 2006).In the context of the MDE, model transformation is based on a set of rules specifying the relationship between the elements of the source model and those of the target model.

Aiming to resolve construction and evolution problems of recent systems, OMG (Object Management Group) has defined a new orientation in 2000, namely MDA (Model-Driven Architecture)(Blanc, 2005). The differences between MDA and the former approach, which is OMA (Object Management Architecture) are the positioning of the latter in a high level of abstraction and the focusing on model approaches rather than object ones. The goal is to concentrate on developing high level of abstraction models and to promote transformational parametric approaches into technical platforms(Bézivin & Blanc, 2002).

The main principle of MDA is to generate platform specific models (PSMs) from platform independent models (PIMs). To this end, MDA relies on technics, such as modeling and models transformation. Furthermore, models transformation is also used in order to define DSLs, a fact that is promoted by MDE.

Models are also at the heart of the simulation approach. As a fact, several simulation environments promote the use of models in order to simulate systems for their behavior verification. In the next section, we propose a study of modeling in simulation environments through the analysis of their common concepts, semantics and modeling methodologies.

2.2 Modeling Systems in Simulation Environments

In order to build models using concepts of simulation environments, a study of their common concepts, semantics and modeling methodologies seems to be essential.

Simulation Modeling Methodologies: They allow to specify the nature of the information produced by the simulation. According to *(Matei & Bock, 2012c)*, there are two main methodologies for modeling simulation,

namely Signal flow modeling and Physical interaction modeling.

• **Signal-flow Modeling:** In this type of modeling, the relationship between system components is unidirectional. The corresponding port variables are designated as inputs or outputs. Therefore, establishing a connection between components involves linking outputs of some components to the inputs of other components, with respect to the following constraints (Matei & Bock, 2012c):

• An input port is linked only to one output port;

• An output port cannot be linked directly to the input port of the same component;

• An output port can be linked to multiple input ports.

Moreover, signal-flow modeling is adapted to the modeling of information movement (signals). In this context, control engineering and signal processing remain action areas for this type of modeling since physical conservation laws are not applied because the same information can circulate to several components unlike physical entities. Among simulation environments using this type of modeling, we mention MATLAB/Simulink®.

• **Physical-interaction Modeling:** The relationship between system components, in this case, is bidirectional, since the actions of a component "A" on a component "B" generate reactions of "B" on "A". As an example, water transmission between two components is affected by maximum capacity and transmission speed of the target component.

Therefore, port variables are not designated as inputs or outputs, but rather as bidirectional. As this type of modeling is adapted to model the movement of physical entities (such as electricity, water, energy, etc.) rather than information, physical conservation laws are thus applied (see Simulation modeling constructs: Link). In this context, Modelicaâ is an example of a simulation environment using this type of modeling.

Basic Constructs of Simulation Modeling Languages: During system modeling, the first step consists on defining the main components of the system and how they interact with each other. Some components may be decomposed further into subcomponents. This decomposition stops when the desired granularity level is reached. The next step in the modeling process is to specify the components in the form of mathematical models and define the relationships that link the different components.

According to *(Matei & Bock, 2012b)* (Matei & Bock, 2012a), modeling in simulation environments is based on four main concepts, organized in a hierarchical form, which are: components, ports, links and subsystems.

• **Component:** Considered as the atomic elements on the basis of which models are constructed, components consist of variables and parameters. The former represent quantities that vary over time, while the latter remain unchangeable during the simulation. Variables and parameters are used in mathematical relationships (equations) to specify the mathematical model describing the behavior of the component.

A mathematical model can be classified as dynamic or static; continuous, discrete or hybrid. A mathematical model is considered as dynamic when time is included in the equations, causing change of variables. On the other hand, a static model is used instead to describe a steady state or equilibrium regime. Variables of a continuous model vary continuously over time, while those of a discrete model change only during specific moments of time. However, models including continuous and discrete components are classified as hybrid.

There are many kinds of equations used in mathematical models, such as differential, algebraic, partial differential and difference equations (Fritzson, 2011). The most used one in simulation tools is differential algebraic equations (DAEs). In the case of a continuous mathematical model, the DAE can be expressed as:

$$f(x(t), x(t), u(t), y(t), t, \theta) = 0$$

$$x(0) = x_0$$

where θ, t, $y(t)$ and $u(t)$ are respectively parameters, time, output and input vectors. Also known as state vector, $x(t)$ is a vector of dependent variables, while $x(t)$ is its derivative. As for f, it refers to a function-valued vector.

In the case of a discrete mathematical model, the DAE can be specified as follow:

$$G(x(k+1), x(k), u(k), y(k), k, \theta) = 0, \quad x(0) = x_0$$

As k is the discrete time, $x(k)$ is updated at discrete time instants.

• **Port:** The interfaces, used by components and subsystems to interact with other components and subsystems, are called ports. This interaction is based on sharing or exchanging variables with other components or subsystems through these ports.

• **Link:** Establishing a connection between two components relies on connecting their ports through links. The latter allow the exchange of energy or information between components. Once the connection is established between components via links, mathematical formulas are generated and thus become part of the entire model behavior. Depending on the adopted modeling methodology, the mathematical formulas generated are subject to certain constraints.

Signal-flow Links: In this type of modeling, variables are

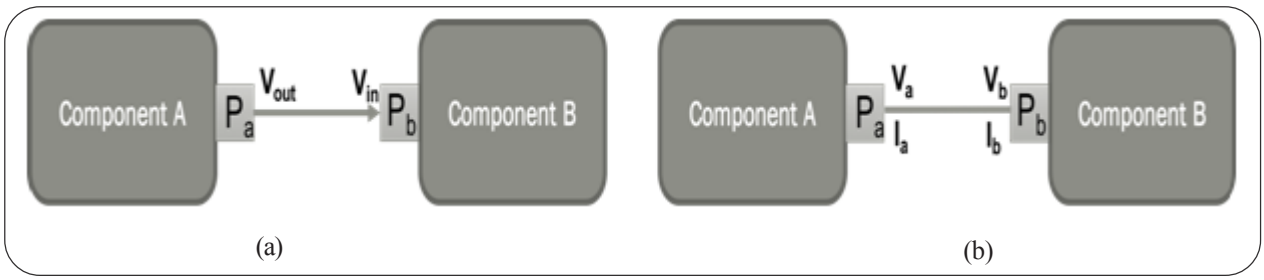


Figure 2. Links depending on the modeling methodology

specified as inputs or outputs. Establishing a link between components relies on connecting the output port of one component to the input port of another component with respect to constraints named earlier. Figure 2.a illustrates a connection between two components in the case of signal flow modeling. In this context, the output port P_a, having a variable V_{out}, is connected to the input port P_b, having a variable V_{in}. Depending on signal-flow modeling semantics, this connection is based on the following assignments:

$$V_{in} := V_{out}$$

The operator := indicates an assignment process since the value V_{in} is affected by the change of the value V_{out}. In other words, the value V_{in} cannot be changed by its component but rather by the changes at the level of the component to which it is linked (in this case, the component A).

Physical-interaction Links: Variables belong to ports whose interaction is qualified as bidirectional. These variables are classified into two categories: conserved and non-conserved variables. The former specifies the rate of movement of physical entities that are not created or destroyed by the system, while the latter describes physical entities in other forms than their movement rates (Matei & Bock, 2012c). Physical-interaction links are governed by the following semantics: the sum of conserved variables values must be zero and the values of non-conserved variables must be equal. Figure 2.b represents a connection between two components in the case of physical-interaction modeling. In this context, the following equations are deduced:

$$V_a = V_b$$
$$I_a + I_b = 0$$

In this case, we are talking about equations and not assignments. Consequently, the values Va and Vb change so as to maintain the aforementioned equality.

• **Subsystem:** A subsystem is a graphical construct that contains 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 other elements through links. The behavior of a subsystem is specified by the behavior of its constituent elements.

3. Simulation Modeling Language

3.1 Global Overview

The main objective of our research work is to study the integration of both SysML modeling and simulation by means of a bidirectional transformation based on the concepts of Model-Driven Engineering (MDE). Our approach consists on the design of a DSL (considered as a PIM) serving as a bridge between SysML models (PIM) and the simulation target environment (PSM) as illustrated in Figure 3. Namely SimulML (Simulation Modeling Language), this language is performed on the basis of common constructs of several simulation environments as well as simulation modeling methodologies (Chabibi, Anwar, & Nassar, 2016). Therefore, the bidirectional transition between SysML models and simulation environment consists on a passage through this intermediate language by using a set of model transformation rules.

Our integration approach is based on SimulML, aDSL that allows linking both SysML and simulation environments tools. SimulML metamodel is built on the basis of the study of common constructs, semantics and modeling methodologies of simulation environments. The definition of SimulML is based on specifying its abstract and concrete syntaxes, and its semantics.

In addition to SimulML definition, this paper proposes a model transformation from this DSL to the simulation environment MATLAB/Simulink in order to illustrate the efficiency of SimulML. This model transformation aims to allow conducting verifications on systems modeled with SimulML.

3.2 Definition of Simulation Modeling Language

SimulML is a language defined in order to bridge the gap between both SysML and simulation environments models. As a fact, it is performed on the basis of common constructs, semantics and modeling methodologies of simulation environments. This section resumes its defining through the specification of its abstract syntax, its semantics and its concrete syntax, respectively. According to (Combemale, 2008), a modeling language can be defined by the tuple $\{AS, CS^*, M^*_{ac}, SD^*, M^*_{as}\}$, where AS is the abstract syntax, CS^* is (are) concrete syntax(es), M^*_{ac} is the set of mappings between AS and M^*_{as}, SD^* is(are) the semantic domain(s) and M^*_{as} is the set of mappings

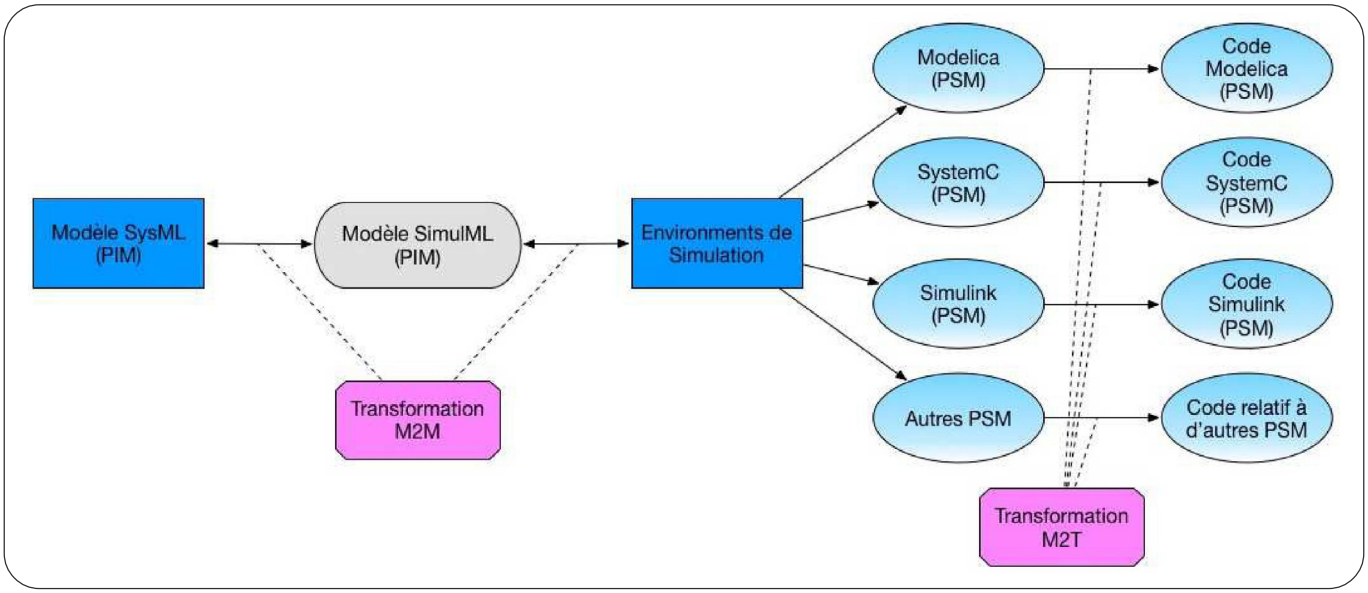


Figure 3. Linking SysML modeling to simulation environments through SimulML

between AS and SD^*.

SimulML Abstract Syntax: The description of a modeling language is based on the specification of its abstract syntax (AS) which allows to express, in a structural way, all its concepts and their relations. In order to specify the AS of a language, several metamodeling languages and environments were created, such as Eclipse-EMF/Ecoreor Kermeta (Steinberg, Budinsky, Paternostro, & Merks, 2008), (Jézéquel & Perrouin, 2008).Concerning the specification of SimulML abstract syntax, we choose the Eclipse-EMF/Ecore. In fact, the Eclipse Modeling Framework (EMF) is a modeling framework and code generation facility for building tools and other applications based on a structured data model. The core EMF framework includes a metamodel (Ecore) for describing models and run-time support for the models, including change notification, persistence support with default XMI serialization, and a reflective API for manipulating EMF objects generically.

Figure 4 illustrates SimulML metamodel. A simulation model is based on four elements, namely: subsystems, components, ports and links. A mathematical model, used to specify the relationship between behavioral variables via mathematical expression(s), describes the behavior of either component, subsystem or the entire system. Parameters and variables take the form of either vector valued or scalar. Components interact with each other through ports. The flow that flows through ports of a component/subsystem allows specifying aspects of the energy or information exchanged with other entities. Moreover, a flow is defined also via flow variables.

However, a port has only one kind of energy or information. The nature of ports depends strictly of the modeling methodology adopted. As a fact, in the case of signal-flow modeling, ports are designated as inputs or outputs. Their correspondent flow variables are based only on non-conserved variables. In the case of physical-interaction modeling, ports are considered as bidirectional and their variables are divided into two categories: conserved and non-conserved variables.

Components ports are connected through links with respect to the following constraints: an input can only be connected to one output; an output cannot be directly connected to the input of the same component; an output can be connected to several inputs. The establishment of the connection between ports of components induces the exchange of energy/information between these components. Nevertheless, this exchange is conformed to semantics of the modeling methodology adopted.

Concerning subsystems, they consist on graphical constructs that contain components and/or subsystems. Furthermore, a subsystem contains ports that interact with others elements through links.

Even though abstract syntax is sufficient to provide a consistent representation of the system since it focuses on the basic concepts of the domain. However, it is insufficient to understand accurately the meaning of the model and how to interpret it unambiguously. In this context, the definition of the modeling language semantics remains essential.

SimulML Semantics: If abstract syntax allows to define the concepts of a language and the relations that exist between them, it does not give the meaning of these concepts, hence the usefulness of the semantics. Indeed, the latter makes it possible to give a precise and unambiguous meaning to the concepts defined in abstract syntax of a language. Semantics is qualified as formal when it is expressed in a mathematical formalism and verifies the concepts definition consistency and

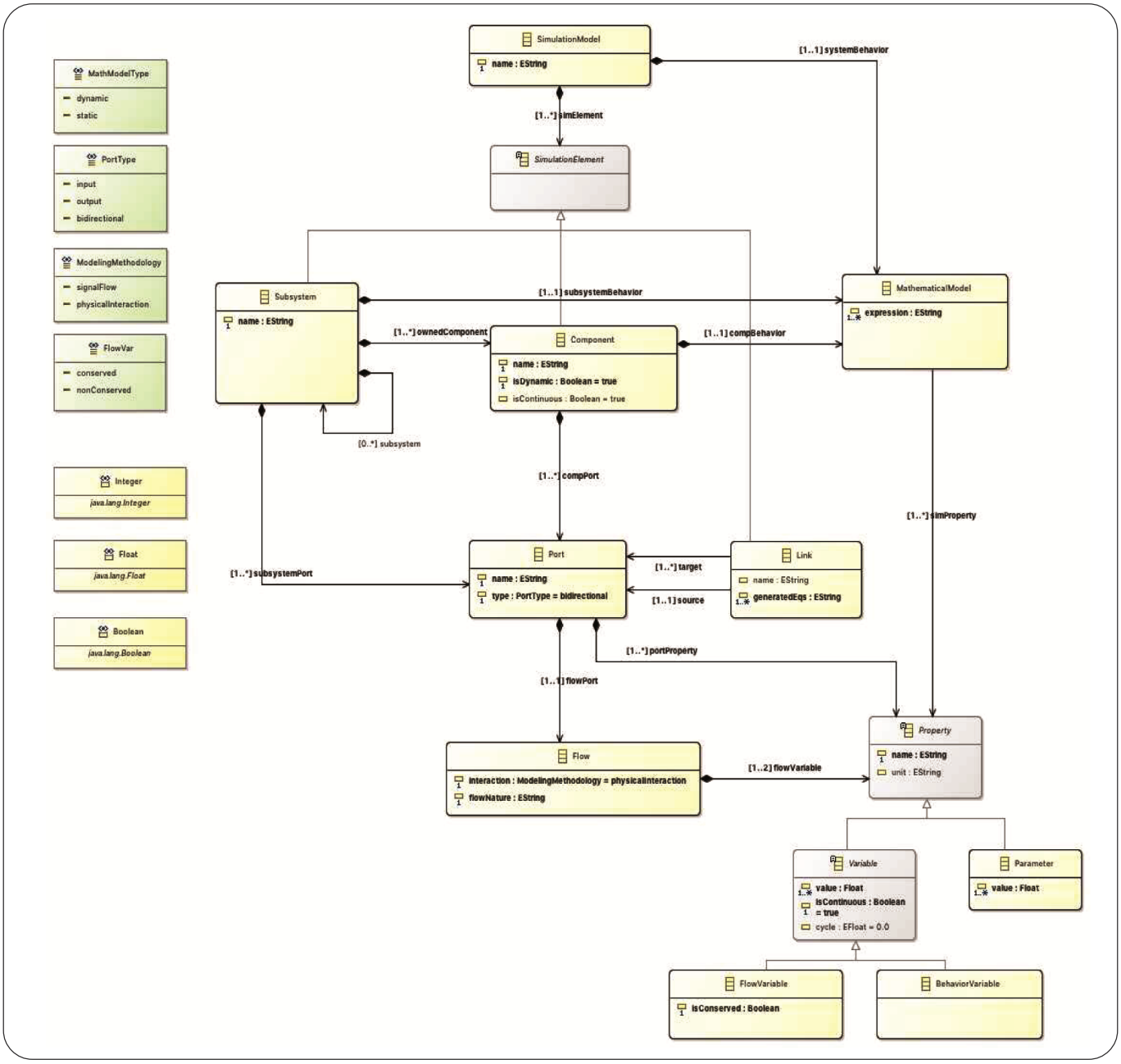


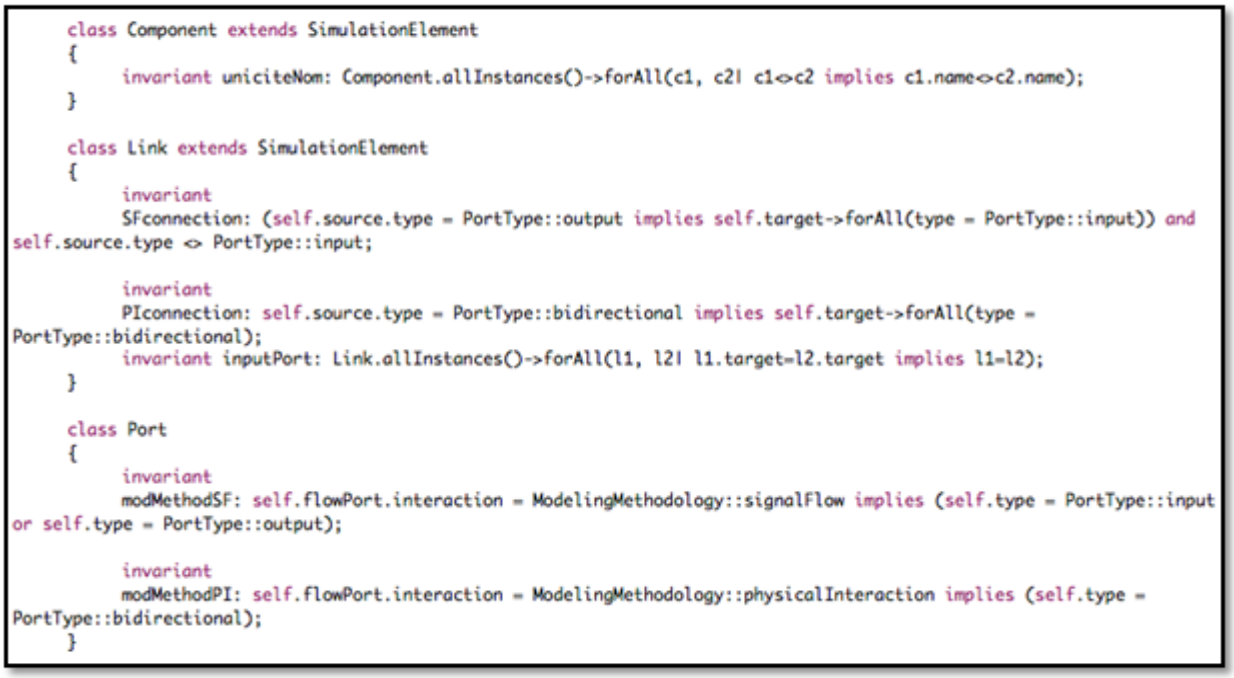
Figure 4. SimulML abstact syntax

completeness. Therefore, the definition of a modeling language semantics consists on defining or precising the semantic domain SD (Figure 1), and specifying one or more mappings M^{*}_{as} between AS and SD.

In the case of SimulML language, some semantics are described directly on the metamodel, by using multiplicities. As a fact, the constraint "a simulation model, subsystem or component behavior is specified by a unique mathematical model" is specified respectively through the associations "systemBehavior", "subsystemBehavior" and "compBehavior", that link the super-classes SimulationModel, Subsystem and Component to "MathematicalModel" super-class of SimulML metamodel. The cardinality of the reference *target* (1..1) allows describing the unicity aspect.

Furthermore, in order to define some properties on the abstract syntax, the metamodel is completed with constraints expressed in OCL. The latter is a functional language based on first-order logic that allows expressing specifications on a UML model.OCL can be used as a specification language of metamodels, which requires good knowledge of both syntax and semantics of modeling elements. Furthermore, well-formedness rules can be added to metamodelsin order to restrict the set of valid instances (Bazex, Bodeveix, Millan, Camus, & Percebois, 2003).

In this context, among several constraints expressed using OCL in SimulML metamodel, we list some examples in Figure 5. The constraint "uniciteNom", that concerns "Component" context, is used to specify that



```
class Component extends SimulationElement
{
    invariant uniciteNom: Component.allInstances()->forAll(c1, c2| c1<>c2 implies c1.name<>c2.name);
}

class Link extends SimulationElement
{
    invariant
    SFconnection: (self.source.type = PortType::output implies self.target->forAll(type = PortType::input)) and
self.source.type <> PortType::input;

    invariant
    PIconnection: self.source.type = PortType::bidirectional implies self.target->forAll(type =
PortType::bidirectional);
    invariant inputPort: Link.allInstances()->forAll(l1, l2| l1.target=l2.target implies l1=l2);
}

class Port
{
    invariant
    modMethodSF: self.flowPort.interaction = ModelingMethodology::signalFlow implies (self.type = PortType::input
or self.type = PortType::output);

    invariant
    modMethodPI: self.flowPort.interaction = ModelingMethodology::physicalInteraction implies (self.type =
PortType::bidirectional);
}
```

Figure 5. Some constraints expressed using OCL in SimulML metamodel

the name of a component within a simulation model is unique.

In the case of "Link" context, there are three examples of specified constraints. The "SFconnection" constraint is used to express that a signal-flow link allows combining an output port to one or several input ports. Concerning the "PIconnection", it stipulates that a physical-interaction link is used to combine bidirectional ports. The "inputPort" constraint reflects the linking semantic stipulating that an input can only be connected to one output.

As for the "modMethodSF" constraint of the "Port" context, it specifies the typeof a port as an input or an output port in the case of signal-flow modeling. Concerning the "modMethodPI" constraint, it states that a port is considered as bidirectional in the case of a physical-interaction modeling.

In order to manipulate the abstract syntax of SimulML language and generate its instances, a concrete syntax of the language is performed.

SimulML Concrete Syntax: The specification of a language concrete syntax (CS) provides the user with an effective way to create models. This specification consists on defining one or more tools, graphic and/or textual, to manipulate the concepts defined in the abstract syntax. As a consequence, the created models (considered as instances) conform to the abstract syntax metamodel.

The definition of a concrete syntax is based on defining one of the mappings of M^{*}_{ac}, illustrated in Figure 1, between AS and CS. Therefore, each concept, specified in the abstract syntax, is describedvia one or more decorations of the concrete syntax. Thus, the manipulation of the defined language consists on manipulating the decorations specified in the concrete syntax.

A model can be presented in different ways according to the expressed needs (language expressiveness, target audience, etc.) (Jézéquel et al., 2012). As a fact, a model can take the form of diagrams or trees if concrete syntaxes are graphic, or the form of a text file if they are textual. The goal is to use the syntax that offers more expressiveness and simplicity during its use. In this context, there are a number of environments for specifying the concrete syntax of a language. Projects such as EMF (Eclipse Modeling Framework), TOPCASED, Sirius, etc., are dedicated to the definition of mainly graphical concrete syntaxes, while EMFTextand Xtext allow to specify textual concrete syntaxes.

Xtext is a framework allowing the development of programming languages and DSL (Domain-Specific Language). A DSL is considered to be a language specific to a trade or a company that follows a defined and compact formalism. The Xtext framework is based on an ANTLR (ANother Tool for Language Recognition) generated grammar as well as on the EMF modeling framework. Xtext covers all aspects of a modern IDE (Integrated Development Environment): parser, compiler, interpreter, and full integration in the Eclipse development environment. As such, it covers the main aspects and functionalities of a traditional IDE. Due to its potentials, Xtext enables developers to have a powerful tool that provides a user-friendly environment for DSL users.

In this section, we describe the implementation of SimulML using the Xtext framework for Eclipse, without

```
SimulationModel returns SimulationModel:
    'SimulationModel'' 'name=EString
    '{'
        'systemBehavior'': ' systemBehavior=SYMBOLE
        'modelElements' '{' elements+=SimulationElement ( ',' elements+=SimulationElement)* '}'
    '}';

SimulationElement returns SimulationElement:
    Subsystem | Component | Link;

MathematicalModel returns MathematicalModel:
    'MathematicalModel'
    '{'
        'expression' '{' expression+=SYMBOLE ( "," expression+=SYMBOLE)* '}'
        'properties' '{' simProperty+=Property ( "," simProperty+=Property)* '}'
    '}';

Subsystem returns Subsystem:
    'Subsystem' ' ' name=EString
    '{'
        ('ownedSubsystems' '{' subsystem+=Subsystem ( "," subsystem+=Subsystem)* '}' )?
        'ownedComponents' '{' ownedComponent+=Component ( "," ownedComponent+=Component)* '}'
        'ownedPorts' '{' subsystemPort+=Port ( "," subsystemPort+=Port)* '}'
        'subsystemBehavior'': ' subsystemBehavior=MathematicalModel
    '}';
```

Figure 6. SimulML grammar implemented in Xtext

providing all the details of this implementation. The first task in Xtext is to write the grammar of SimulML using an EBNF-like syntax (Extended Backus-Naur Form). A part of this grammar is illustrated in Figure 6. Starting from this grammar, Xtext generates an ANTLR parser in addition to the abstract Syntax Tree (AST). The latter takes the form of an EMF model. In this context, Xtext connects the EMF model representing the AST with the textual form of the program in the text editor, and maintains their automatic synchronization. As a consequence, the program can be modified by acting on the model representing it, without manipulating its textual form in the editor.

The code generation phase is handled by Xtext via special tools. This code generation allows evaluating the described grammar through an Eclipse editor. It is to note that the textual specification in the editor is conform to the specified grammar via Xtext. As a consequence, this editor will permit to build SimulML models in a textual form. As a fact, it will allow describing a given system using common constructs, semantics and modeling methodologies of simulation tools.

As previously stated, the purpose of SimulML language is to describe systems using common constructs and modeling methodologies of several simulation environments. Therefore, the SimulML grammar expressed with Xtext, whose a part is illustrated in Figure 6, is specified in order to define a system specified as a simulation

SimulML construct	**Code generation**
Simulation model	**SimulationModel***name* { systemBehavior : … modelElements {…, …}. } ;
Subsystem	**Subsystem***name* { ownedSubsystems {…, …} ownedComponents {…, …} ownedPorts {…, …} subsystemBehavior : … } ;

Component	**Component***name* { isDynamic : … isContinuous : … componentBehavior : … ownedPorts {…, …} } ;
Link	**Link***name* { source … target (…, …) } ;
Port	*PortType***Port***name* : { flow : … } ;
Mathematical model	**MathematicalModel** { expression {…, …} properties {…, …} } ;
Flow	*ModelingMethodology* { refersTo : … flowVariable : {…, …} } ;
BehaviorVariable	**PortVariable***name* { isContinuous : … cycle : … unit … value (…, …) } ;
FlowVariable	*name* { isConserved : … isContinuous : … cycle : … unit : … value : (…, …) } ;
Parameter	**Parameter***name* { unit : … value : (…, …) } ;

Table 1. SimulML operators and attributes

model (*SimulationModel*) whose characteristics are its name, its behavior (*systemBehavior*) and the simulation elements it contains (*modelElements*). These are either a subsystem, a component, or a link. Subsystem is specified by its name, its behavior (*subsystemBehavior*), its components (*ownedComponents*) and the subsystems

(*ownedSubsystems*) it contains, as well as the ports through which it communicates with the other subsystems and components (*ownedPorts*). A port is defined by its type (*PortType*), its name and the flow that flows through it.

A flow refers to the nature of substances that cross a link. It is specifiedthrough the modeling methodology adopted (*ModelingMethodology*), the nature of the constituent substances (*refersTo*) and its corresponding variables (*FlowVariable*). In addition, a flow variable is defined through the physical conservation aspect (*isConserved*), its variationthrough time (*isContinuous*), the change cycle of values (*cycle*) in the case of a discrete variable, its unit (*unit*) and the vector of its values (*value*). As for the link, it is described through its name, the output port (*source*) and all the input ports (*target*).

As for components, they are described through their name,the characteristics of their variation over time (*isDynamic*/*isContinuous*), their ports (*ownedPorts*) with which they are connected to other components and subsystems, and their behavior (*componentBehavior*). Moreover, the behavior is used to specify the mathematical model (*MathematicalModel*) of the component, subsystem or simulation model.This mathematical model is characterized by the mathematical expression (*expression*) describing the behavior and properties included in this expression. Since they are exclusively port-related properties whose purpose is to describe behavior, these properties consist of variables or parameters. A parameter is described via its name, unit (*unit*) and the vector of its values (*value*), while a variable is specified by its variation over time (*is Continuous*), the cycle of change of values (*cycle*) in the case of a discrete variable, its unitand the vector of its values (*value*).

To sum up, Table 1 summarizes SimulMLconstructs, as well as their corresponding grammar generated through the compilation of Xtext file thatspecifies its concrete syntax. The code generation phase allows to describe systems through an Eclipse editor using common constructs and modeling methodologies of simulation environments. In this context, Figure 7 depicts a SimulMLtextuel modeling example that consists on the specification of a temperature sensor using the generated Eclipse editor.

```
SimulationModel Heating_System
{
    systemBehavior: Tf=Ti+(R*I^2*t)/4186
    modelElements
    {
        Component Temperature_Sensor
        {
            isDynamic: true
            componentBehavior: MathematicalModel
            {
                expression
                {
                    Rt=R0*exp(beta*(1/T-1/T0))
                }
                properties
                {
                    PortVariable activate_out
                    {
                        isContinuous: true
                        unit: signal
                        value: (1.0)
                    },
                    PortVariable T
                    Parameter beta
                    Parameter T0
                    Parameter R0
                }
            }
            ownedPorts
            {
                input Port temperature_in
                {
                    flow: signalFlow
                    {
                        refersTo: realSignal
                        flowVariable:
                        {
                            T_in
                            {
                                isConserved: false
                                isContinuous: true
                                value: (1000)
                            }
                        }
                    }
                },
                output Port cmd_out
            }
}
```

Figure 7. An example of SimulML textual modeling

SimulML-MATLAB/Simulink Models Transformation

In order to illustrate SimulML use and efficiency in our integration approach, we perform a model transformation from this DSL to MATLAB/Simulink simulation environment. The model transformation consists on a Model-To-Text (M2T) approach. Therefore, among the various languages that allows this type of transformation (i.e JET, Xpand, …), we use Acceleo language which is an Eclipse implementation of the OMG MOF2Text transformation Language (MTL)(OMG, 2008).

The M2T proposed approach is based on MATLAB code generation from SimulML models. The main elements that allow performing this transformation based on Acceleo tool are illustrated in Figure 8. The principle of the M2T approach consists on using Acceleo templates (.mtl) in order to define transformation rules performed on SimulML source models. The Acceleo templates and SimulML source models are used by an Eclipse generator in order to generate the corresponding MATLAB source code (.m) conformed to defined transformation rules.

In order to perform the M2T transformation from SimulML to MATLAB, a manipulation of constructs defined in SimulML metamodel is required. As a fact, a good knowledge of this metamodel, and the relationship between the metamodel constructs and concrete syntax is necessary to implement the model transformation.

The definition of Acceleo templates is mainly based on specifying transformation rules from SimulML to Simulink by defining mapping between these two languages' constructs. Defined mapping for a specific domain can be used in the development of several applications in the same domain. Table 2 summarizes the mapping performed between SimulML and Simulink constructs.

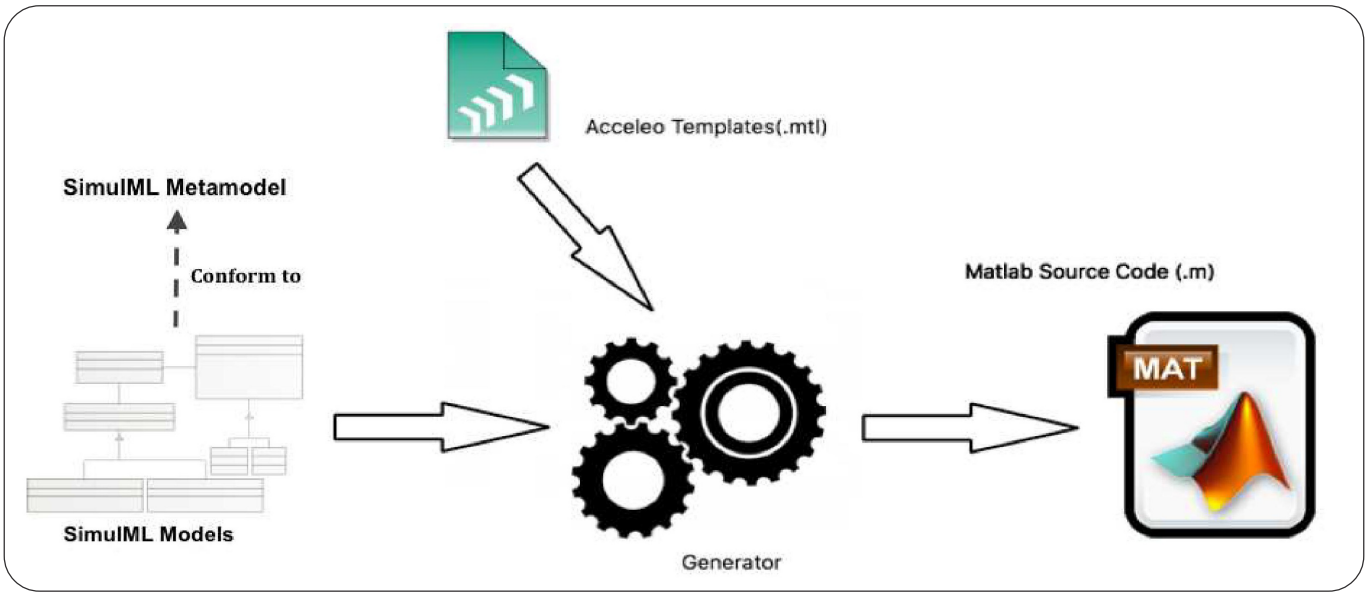


Figure 8. MATLAB code generation approach

SimulML	Simulink	Note
Simulation Model	Block (system)	SimulationModel turns into the complete Simulink system
Subsystem	Subsystem block	Each part of a SimulML subsystem is modeled with a Simulink subsystem block
Port	Input/Output block	SimulML ports are modeled with Simulink Input and Output blocks
Mathematical Model	Dynamic equation	SimulML mathematical models are modeled in Simulink using dynamic equations
Link	Line	Used to connect ports, SimulML links are modeled in Simulink by lines

Table 2. SimulML and Simulink constructs mapping

```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/emf/2002/Ecore')]
[import operations/]


[template public generateElement(b : Model)]
[comment @main/]
[for (p: Element | self.ownedElement)]
     [if (p.oclIsTypeOf(Package)]
          [for (e: Class | p.ownedElement)]
               [generateBlock(e, b)/]
               [generateSimulationModel(e, b)/]
          [/for]
     [/if]
[/for]
[/template]
```

Figure 9. Example of Acceleo template

Using this mapping, Acceleo templates are created (Figure 9). These templates consist on texts containing spaces in which source model information will be placed. The MATLAB code generator, illustrated in Figure 8, is based on both SimulML models and Acceleo templates. The former is sent as input to the code generator. The latter generates the correspondent of each SimulML source model construct as it has been defined in Acceleo templates. As a consequence, an executable MATLAB code is generated in the form of script files (.m). This files are used to obtain Simulink models through MATLAB application.

In the next section, we present SimulML textual editor, through the study of a practical case whose objective is SimulMLmodeling of water heating system in a coffee machine. Furthermore, we illustrate the passage from SimulML to MATLAB/Simulink through the generation of Simulink models, obtained via an M2T defined transformation.

4. An Illustrative Example

In order to illustrate modeling with SimulML language, we have created an instance of SimulML simulation model that describes the water heating system contained in a coffee pod machine. The operating principle of the latter can be summarized as follows: When the user presses the coffee preparation button, a water pump starts pumping water from the tank through a flowmeter. Once at the water pump, water is compressed and fed to the solenoid valve and then to the boiler. Water temperature in the boiler is controlled by a temperature sensor attached to it and the value is generally between 98° and 110°C. The water coming out of the boiler passes through a compression chamber where the pod is placed. Water infused through

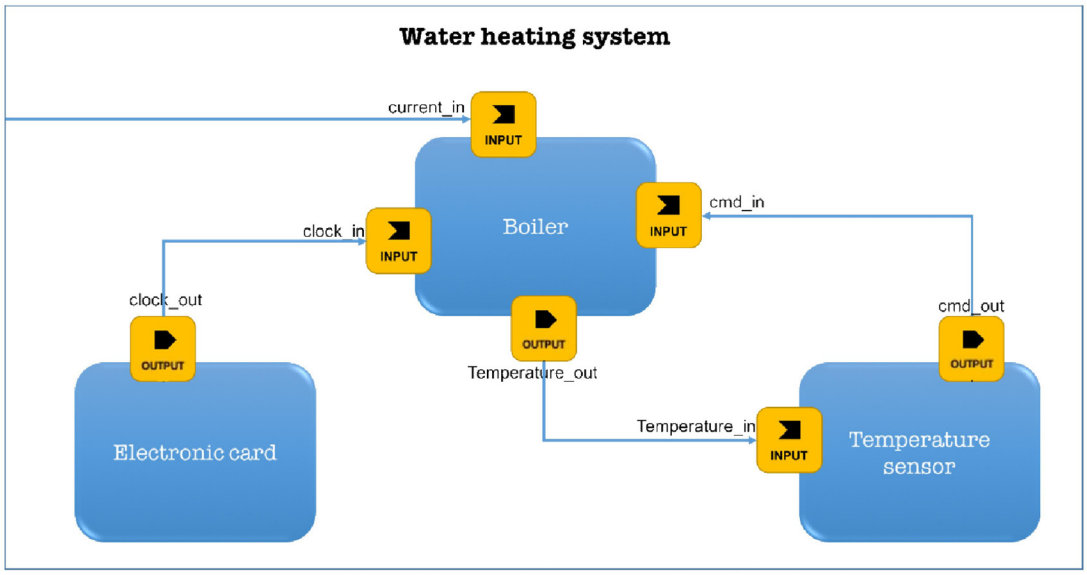


Figure 10. Modeling of the water heating system of a coffee pod machine

the pod and coffee starts to flow. When the flowmeter has finished measuring the water needed to fill the requested cup, it sends a signal to the solenoid valve to shut off the water and the vibration pump stops pumping.

The water heating system of the coffee pod machine is composed of three components: the boiler, the temperature sensor and the electronic card. These three components are linked together through a signal-flow linking.Figure 10 illustrates the linking between these components as well as ports used to exchange information through links. As for Figure 11, it depicts SysML IBD (Internal Block Diagram) of the studied system.

The coffee machine boiler is used to heat water increasingly with time through the use of a resistance.Figure 12depicts summarily some requirements related to water heating. Its operating principle is based on the Joule's law: "the power dissipated by Joule effect in an ohmic conductor is proportional to the resistance of the conductor and to the square of the intensity of the current flowing through it".

We often cite the formula $P = R.I^2$ although formally it is valid only if R and I remain constant, and in the case of conductors whose resistance does not depend on the temperature.

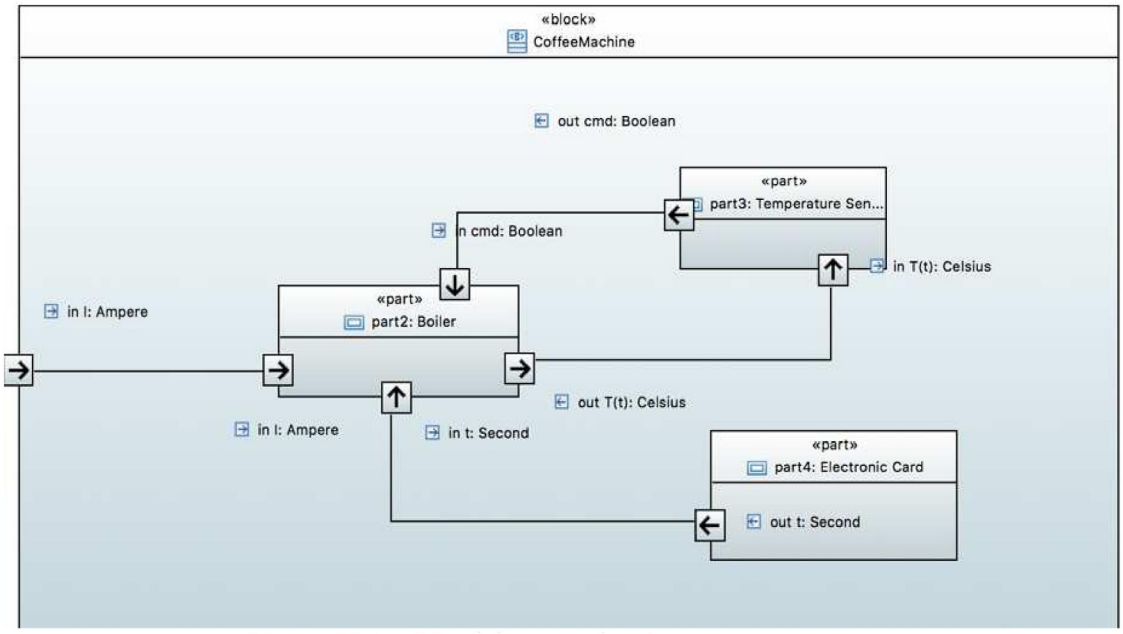


Figure 11. An IBD of the water heating system

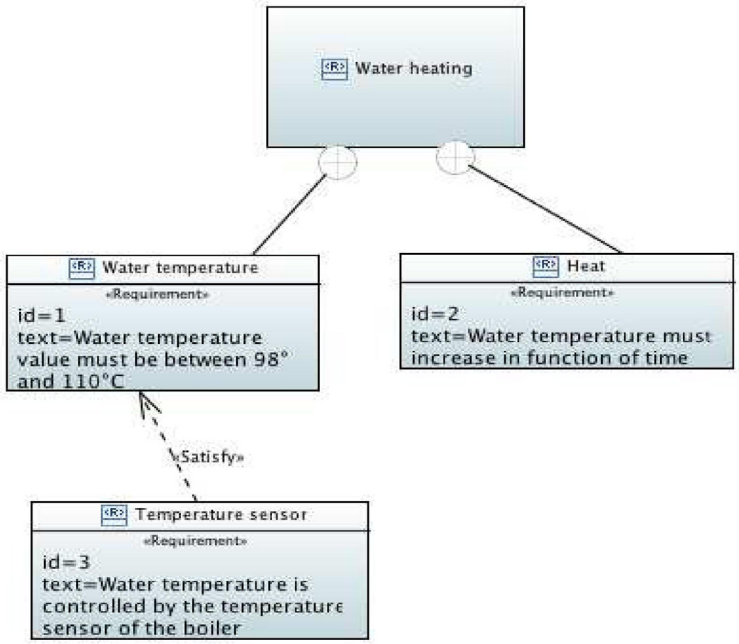


Figure 12. Requirement diagram of the coffee machine boiler

Power is the amount of energy consumed per unit of time. By definition, we can calculate the total amount of energy dissipated by Joule effect by multiplying the power (W) by the phenomenon duration.

$$W = P.t$$
$$\mathrm{W} = R.I^2.t$$

The temperature of 1Kg of water needs 4186J to be raised by 1°C. As a fact, to heat 1Kg of water from T_i (initial temperature = 20°C) to T_f (final temperature <= 98°C), the formula becomes:

$$W = (Tf - Ti).4186\,(Kg/J).\,(1kg)$$

$$R.I^2.t = (Tf - Ti).4186$$

$$Tf(t) = \frac{R.I^2.t}{4186} + Ti \qquad (1)$$

I, R and *t* represent respectively current intensity, resistance and time.

Thus, the eq. (1) describes the behavior of the boiler component. It is important to note that the created simulation model instance, describing the heating system of the coffee pod machine, is based on signal-flow modeling since it concerns the description of movement of information (signals).

Figure 13 shows a part of SimulMLtextual modeling of the coffee pod machine heatingsystem. We can notice that this system is described conforming to the grammar defined using Xtext. Therefore, the editor provides facilities concerning syntax highlight in terms of font, color, and style for comments, keywords, numbers, and Strings.

The coffee pod machine heating system described in the editor is specified according to a signal-flow modeling. The study of this system was concerning two main components: boiler and temperature sensor. As for the electronic card, its principle consists, in the context of the heating system, on communicating time evolution to the boiler in order to ensure a proportional water heating in function of time evolution.

The boiler communicates with the external environment through four ports. The input port "clock_in" that contains the variable "time" is used by the boiler in order to heat water in function of time evolution. As for the input port "current_in" that contains the variable "intensity", it allows to supply the boiler with the electrical current. The input port "cmd_in" that contains the variable "activate_in" is used to stop water heating by the boiler. Concerning the output port "temperature_out" that owns the variable "*Tf*", it serves to communicate the water temperature to the temperature sensor. Furthermore, the boiler behavior is described by eq. (1), specified as a dynamic continuous mathematical model. The latter contains the variables defined on ports in addition to two parameters, named "*R*"

```
SimulationModel Heating_System
{
    systemBehavior: Tf=Ti+(R*I^2*t)/4186
    modelElements
    {
        Component Temperature_Sensor
        {
            isDynamic: true
            componentBehavior: MathematicalModel
            {
                expression
                {
                    Rt=R0*exp(beta*(1/T-1/T0))
                }
                properties
                {
                    PortVariable activate_out
                    {
                        isContinuous: true
                        unit: signal
                        value: (1.0)
                    },
                    PortVariable T
                    Parameter beta
                    Parameter T0
                    Parameter R0
                }
            }
            ownedPorts
            {
                input Port temperature_in
                {
                    flow: signalFlow
                    {
                        refersTo: realSignal
                        flowVariable:
                        {
                            T_in
                            {
                                isConserved: false
                                isContinuous: true
                                value: (1000)
                            }
                        }
                    }
                },
                output Port cmd_out
            }
        },
        Component Boiler
        Link temperature
        {
            source: temperature_out
            target: (temperature_in)
        },
        Link cmd
    }
}
```

Figure 13. A part of SimulML modeling of water heating system

(Resistance) and "*Ti*" (initial temperature).

As for temperature sensor, its behavior is described through the following dynamic continuous mathematical model (eq. 2):

$$R_T = R_0 * exp\left(\beta\left(\frac{1}{T} - \frac{1}{T_0}\right)\right) \qquad (2)$$

R_T is the resistance that corresponds to a temperature T. The resistance value R_0 is considered as a reference that is equal to 10.000 ohms, conforming to the initial temperature $\mathrm{T}_0 = 298$ Kelvins. β is a constant value (equals to 3435 Kelvins). The temperature sensor has two ports: an input port named "temperature_in" that contains the variable T, and an output port "cmd_out" containing the variable "activate_out".

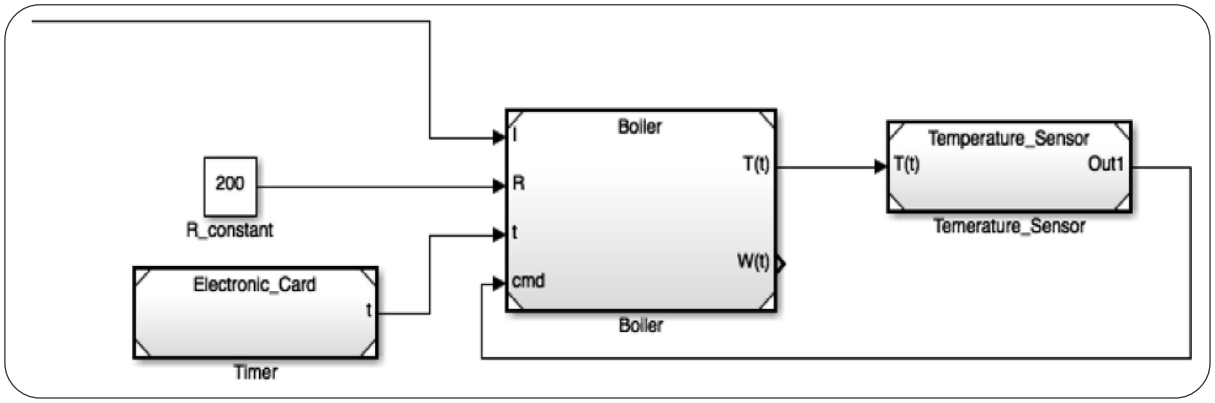


Figure 14. Simulink model generated from SimulML source model

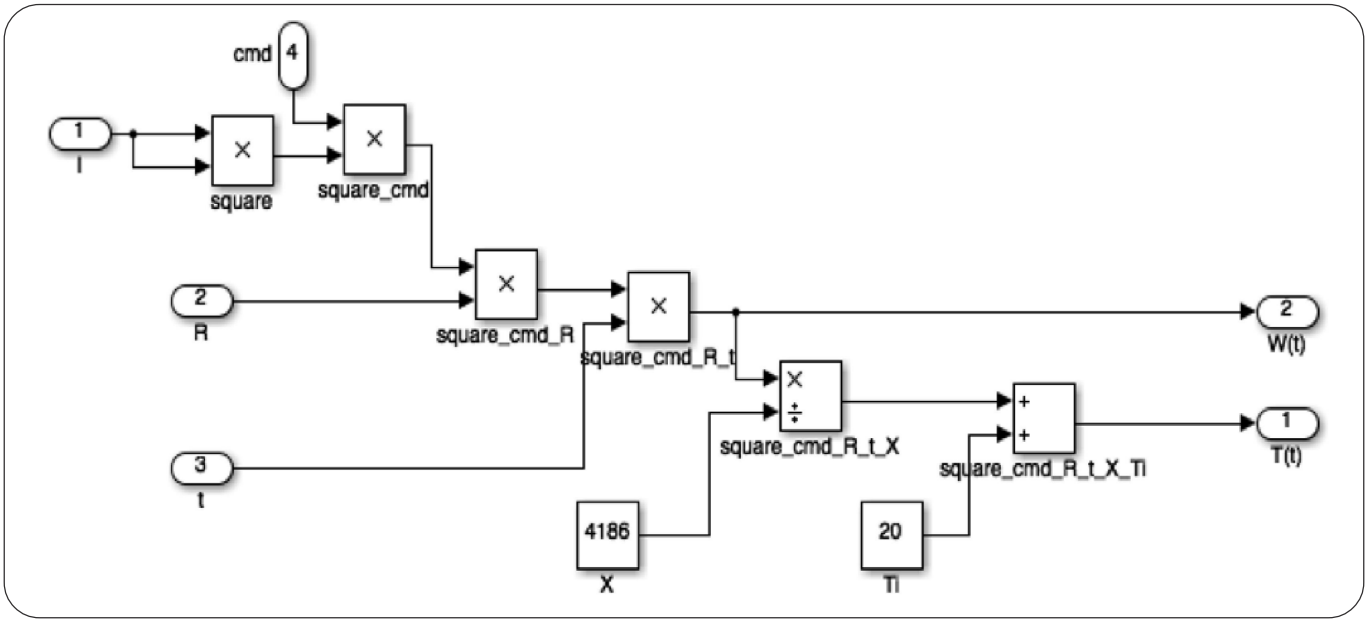


Figure 15. Simulink model obtained from the boiler mathematical model

In order to apply the M2T transformation process presented earlier, SimulML models of the studied system was sent as an input of the MATLAB code generator. The latter generated script files (.m) containing executable MATLAB code. Used in a MATLAB application, the obtained script files had been executed in order to perform the correspondent Simulink models. Thus, Figure 14 and Figure 15 illustrate Simulink models automatically obtained from SimulML source model. In this context, it is to note that Figure 15 is performed through the specification of the boiler mathematical model that defines the equation linking several variables and parameters of this component. This mathematical model was submitted to transformation rules defined in Acceleo templates in order to generate the Simulink model depicted in Figure 15.

After obtaining Simulink models via M2T transformation process, we conducted experiences in order to study water temperature variation through time. Aiming to verify requirements specified in Figure 12, this study was being based on the boiler mathematical model. By specifying initial data as initial water temperature estimated to 0°C and the studied coffee machine power that is equal to 800W, we were able toobserve the final water temperature variation through time via the simulation illustrated in Figure 16. As a consequence, it appeared that the temperature value did not overstep 98°C. This observation allowed inferring that the temperature sensor has prevented the boiler to heat water beyond the permitted values.

To sum up, this paper resumes the definition of SimulML through the specification of its abstract and concrete syntaxes, and its semantics. SimulML concrete syntax is performed via Xtextin order to create a textual editor allowing to build SimulML models. As a consequence, we defined a language that has permitted modeling a system using common constructs, semantics and modeling methodologies of simulation environments. Furthermore, an M2T transformation process from SimulML to MATLAB/Simulink was defined in order to prove the use and efficiency of this DSL, and to

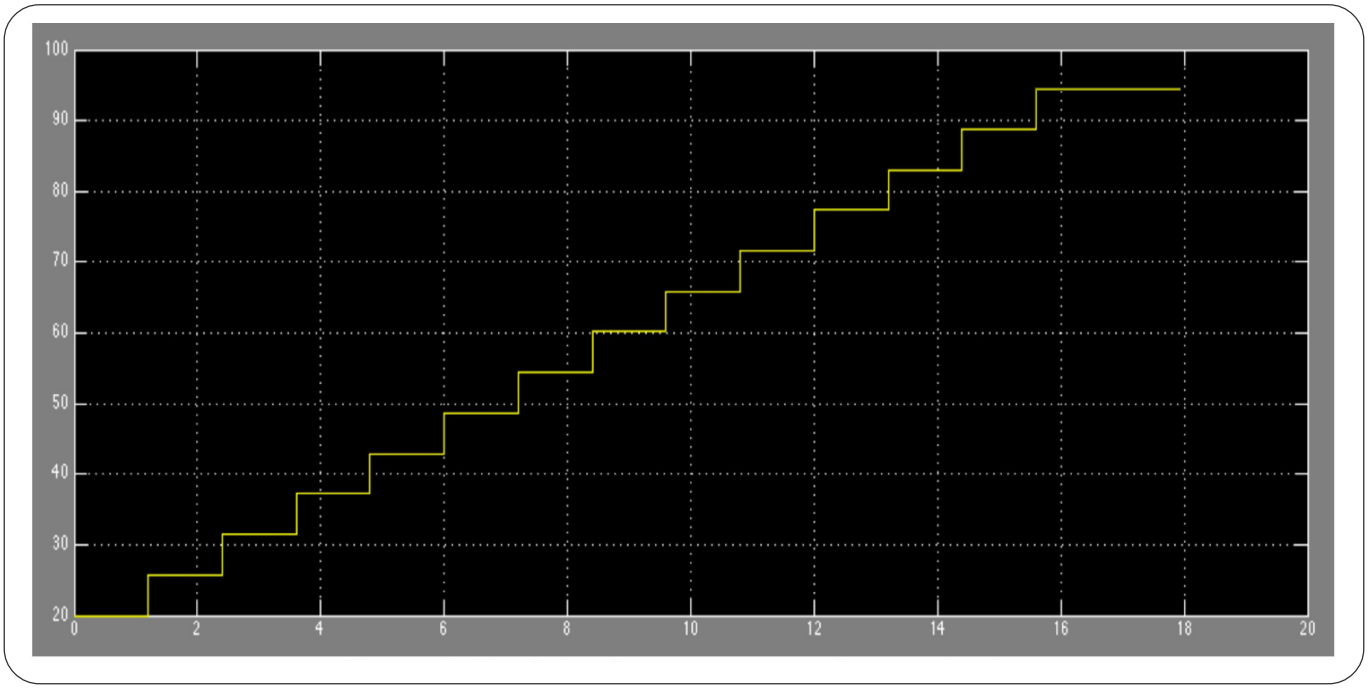


Figure 16. Simulation for temperature values monitoring

demonstrate its adaptability to be linked to several simulation environments as it had been specified on the basis of their common constructs, semantics and modeling methodologies.

5. Discussion and Conclusion

This paper presents the Simulation Modeling Language, built in the basis of a study of the main common constructs, semantics and modeling methodologies of simulation tools.

Thus, SimulML allows creating simulation model of a given system. This model consists on 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 the interaction between them. The behavior of a component is described through mathematical models based on component variables and parameters. Connecting components refers connecting their ports through links. The resulting interaction between components induce the behavior of the system. Nevertheless, the interaction between components depends on the semantics of the modeling methodology adopted.

Moreover, an M2T transformation approach allowing to link SimulML to MATLAB/Simulink is defined in this paper. The model transformation consists on a MATLAB code generator that takes as inputs SimulML model in addition to Acceleo templates where transformation rules are defined. As an output, the code generator generates MATLAB script files that are used in a MATLAB application in order to obtain Simulink model which are conformed to SimulML source model.

The Simulation Modeling Language is the major tool in our integration approach. As a fact, SimulML represents our intermediate language in order to link SysML modeling to simulation environments (i.e. Modelica, etc.) through an approach that is compliant with MDE principles as illustrated in Figure 1.

To this end, we intend to link SimulML to both SysML and a chosen simulation environment through the process of models transformation in order to ensure a bidirectional path between these metamodels. All these elements (i.e. metamodels, model transformations, etc.) are to include in a model-driven integration approach, based on MDE principles, that ensures bidirectional transformation between simulation environments and SysML in order to take advantage of the potential offered by these two approaches.

References

[1] Rahman, Abdul., M. A., Mizukaw, M. (2013). Model-Based Development and Simulation for Robotic Systems with SysML, Simulink and Simscape Profiles. *International Journal of Advanced Robotic Systems*. https://doi.org/10.5772/55533

[2] OMG, A. S. (2008). *About the MOF Model to Text Transformation Language Specification Version 1.0* (No. formal/2008-01-16). Object Management Group. Retrieved from https://www.omg.org/spec/MOFM2T/1.0/

[3] Arunachalam, S., Zalila-Wenkstern, R., Steiner, R. (2008). Environment mediated Multi Agent Simulation Tools –A Comparison (p. 43–48). Presented at the Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on,

Venice: IEEE. https://doi.org/10.1109/SASOW.2008.44

[4] AS MDA. (2005). Action Spécifique CNRS sur l'Ingénierie Dirigée par les Modèles, *1.1.2*, 1–16.

[5] Bazex, P., Bodeveix, J.-P., Millan, T., Camus, C. L., Percebois, C. (2003). Vérification de modèles UML fondée sur OCL. In *ResearchGate* (p. 185–202). Retrieved from https://www.researchgate.net/publication/220764399_Verification_de_modeles_UML_fondee_sur_OCL

[6] Bézivin, J., Blanc, X. (2002). MDA/ : VERS UN IMPORTANT CHANGEMENT DE PARADIGME EN GENIE LOGICIEL. Retrieved from http://www.devreference.net/

[7] Blanc, X. (2005). *MDA en action/ : Ingénierie logicielle guidée par les modèles*. Paris: Eyrolles.

[8] 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.

[9] 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, Agadir, Morocco.

[10] Chabibi, B., Douche, A., Anwar, A., & Nassar, M. (2016). Integrating SysML with Simulation Environments (Simulink) by Model Transformation Approach (pp. 148–150). Presented at the 2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE) (2016), Paris. http://doi.ieeecomputersociety.org/10.1109/WETICE.2016.39

[11] 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. Retrieved from http://tel.archives-ouvertes.fr/tel-00321863/

[12] Ereeþ, S., Kuruoðlu, E., Moralý, N. (2013). An Application of Analytical Hierarchy Process for Simulation Software Selection in Education Area, 60–70. https://doi.org/10.5923/j.fs.20130302.03

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

[14] Fritzson, P. A. (2011). *Introduction to modeling and simulation of technical and physical systems with Modelica*. Hoboken, N.J: Wiley/ : IEEE Press.

[15] Jerry, B. (2005). *Discrete-event system simulation*. Pearson/Prentice Hall.

[16] Jézéquel, J.-M., Combemale, B., Vojtisek, D. (2012). *Ingénierie Dirigée par les Modèles/ : des concepts a la pratique*. Ellipses Marketing.

[17] Jézéquel, J.-M., & Perrouin, G. (2008). Vers des lignes de produits flexibles. Apports de l'ingénierie dirigée par les modèles à la dérivation de produits. *L'objet*, (14/2008), 33–45. https://doi.org/10.3166/obj.14.3.33-45

[18] Jouault, F. (2006). *Contribution à l'étude des langages de transformation de modèles* (Informatique). Université de Nantes, UFR Sciences & Techniques.

[19] Matei, I., Bock, C. (2012a). *An Analysis of Solver-Based Simulation Tools*. Systems Integration Division Engineering Laboratory: National Institute of Standards and Technology. Retrieved from http://www.nist.gov/manuscript-publication-search.cfm?pub_id=909924

[20] Matei, I., Bock, C. (2012b). *SysML Extension for Dynamical System Simulation Tools*. US Department of Commerce, National Institute of Standards and Technology. Retrieved from http://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7888.pdf

[21] Matei, I., Bock, C. E. (2012c). *Modeling Methodologies and Simulation for Dynamical Systems*. National Institute of Standards and Technology. Retrieved from http://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7875.pdf

[22] Object Management Group. (2012). *OMG Systems Modeling Language (OMG SysML™)* (No. 1.3). Retrieved from http://www.omg.org/spec/SysML/1.3/

[23] Seidewitz, E. (Sept.-Oct.). What models mean. *IEEE Software*, 20 (5) 26–32. https://doi.org/10.1109/MS.2003.1231147

[24] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E. (2008). *EMF: Eclipse Modeling Framework* (2nd Revised edition). Upper Saddle River, NJ: Addison-Wesley Professional.

Author's Biography

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.