Towards V&V suitable Domain Specific Modeling Languages for MBSE: A tooled approach

Blazo Nastov, Vincent Chapurlat, Christophe Dony, François Pfister

Towards V&V suitable Domain Specific Modeling Languages for MBSE: A tooled approach

Blazo NASTOV
LGI2P
Parc scientifique Georges Besse
30035 Nîmes CEDEX 1 – France
Blazo.Nastov@mines-ales.fr

Vincent CHAPURLAT
LGI2P
Parc scientifique Georges Besse
30035 Nîmes CEDEX 1 – France
Vincent.Chapurlat@mines-ales.fr

Christophe DONY
LIRMM
161 rue Ada
34392 Montpellier – France
dony@lirmm.fr

François PFISTER
LGI2P
Parc scientifique Georges Besse
30035 Nîmes CEDEX 1 – France
Francois.Pfister@mines-ales.fr

Abstract. Considering Model-Based Systems Engineering (MBSE) principles and needs, this article focuses on the design of Domain Specific Modeling Languages (DSMLs) aiming to link modeling, verification and validation (V&V) activities. The goal is to ease the work and increase the freedom and autonomy of experts from various domains in the process of creating and managing system models then to supply experts involved in analyses and decision making processes with models characterized by highest level of confidence. This paper introduces and illustrates a tool-equipped approach, named "xviCore", that provides MBSE experts with natively verifiable, executable and interoperable DSML named "xviDSML", i.e., DSML that can be formally and directly checked and simulated requesting no huge efforts or skills.

Introduction

Model-Based Systems Engineering (MBSE) is *the collection of related processes, methods and tools used to support the discipline of Systems Engineering* in a "model-based" or "model-driven" context (ISO/IEC, 2008) (Estefan, 2008). MBSE is based then on the creation and the management of various models of a System of Interest (SoI). These models are designed to reach experts' objectives, e.g., understanding, performance analyzes and expected behaviors or '–ilities' (de Weck et al., 2012). Each model focuses on a given aspect of a SoI (functional, logical, physical or behavioral) or a given point of view (user, designer, manager…). Finally, it is used as a base for decision making. It is therefore imperative, prior to any decision (e.g., architectural choice), to proceed model verification and validation (*V&V*) activities. In MBSE context, verification activities aim to demonstrate the consistency, correctness and conformity of the model to its modeling language, modeling rules and modeling patterns, i.e., to demonstrate that a model is correctly built. In the same way, validation activities aim to demonstrate, as much as possible, the relevance and fidelity of a previously verified model, to represent the SoI as expected thanks to stakeholders' requirements, i.e., to demonstrate with a sufficient level of confidence that a model is 'good' for designers' purposes. V&V activities must consider each of the SoI models, taken them first separately, then pieced together with the other models of the same SoI, providing a more complete and suitable representation of it. The goal is then to demonstrate the mutual coherence of SoI models, as well as their adequacy and global fidelity to the SoI to support the designers' objectives with an assured level of confidence and excellence. All these activities request then to define and use Domain Specific Modeling Languages (DSML) that support and facilitate designers when performing V&V activities. This paper presents a tool-equipped approach that aims to reach expected level of confidence in the MBSE context, all along design processes during which various and heterogeneous DSMLs have to be used, from requirements to

architectural engineering. This method, named "xviCore", provides MBSE experts involved in these processes with natively verifiable and an executable DSML, i.e., a DSML that can be formally checked and simulated with no huge efforts from these experts.

This paper is structured as follows. Section 2 explains the motivation of the paper, identifying current issues. Section 3 presents related works on specifying behavior and formal properties in the realm of DSML. Section 4 describes our tool-equipped approach xviCore. A short case study example demonstrates the approach's applicability on well-known MBSE languages in Section 5 before concluding.

Motivation

The basic principles on which a DSML is based are its *syntax* and its *semantics* (Kleppe, 2007). A DSML composed solely of its syntaxes (an abstract and a concrete syntax) is operational for the creation of models proposing various modeling concepts, relations and attributes that are requested to represent the SoI considering aspect and/or designer's point of view. However, achieving V&V requests to reach two goals discussed hereafter.

First, DSMLs generally lack semantics specification (Harel and Rumpe, 2004). The taxonomy of semantics proposed in (Combemale et al., 2009) highlights two different kinds of semantics. One, named *static semantics*, describes DSML's concepts meaning (abstract and concrete syntaxes), and behavior independent structural constraints (pre and post conditions, invariants, etc.). The other, named *dynamic semantics* specifies DSML behavior as a set of rules for each concept and / or each relation composing the DSML. The behavior of a modeled SoI is then represented thanks to these DSML's elements behaviors. For instance, the eFFBD core element *Function* has a behavior describing an input/output transformation of *Items* flows under the control of triggers flows. A function F can perform the expected transformation when trigger flows (if they exist) are present, input Items are available and previous function (if they exist) put in sequence are ended. Then, available resources are reserved and the transformation of energy, material and / or data can begin, providing continuously or after a given duration, output items, unlocking finally reserved resources. Last, any function being decomposable into sub-functions, the behavior of a decomposed function F can be then described at two levels of detail: the transformation provided by F without any details of transformation provided by its sub-functions or the global and more detailed behavior putting in light the role of each of its sub-functions that interact. Such behavior allows logically model simulation, i.e., to execute any eFFBD model in our case. The first goal is then to make a DSML executable.

Second, a DSML as well as models created in conformity with this DSML should respect a set of properties concerning either its structure (syntax) or behavior (dynamic semantics). Hereafter, a property is *a provable or evaluable (i.e. quantifiable or qualifiable) characteristic of an artefact* (i.e., a model M of S built for achieving a design objective) *that translates all or part of stakeholder expectations to be satisfied by this artefact* (Chapurlat, 2014). We promote the use of a formal property modeling language to deploy a formal V&V strategy based on formal property proof, as it is today used successfully in other domains (Mahfouz, 2013). So, the second goal is to make a DSML or any model created by this DSML, "directly provable". Properties can be checked based on the structure and the behavior of the DSML without any model transformation mechanism as used classically (Bérard et al., 2013).

This work aims then to define and develop a tool-equipped method that support simultaneously DSML designers' and model designers' V&V activities including specification and checking of the syntaxes, of behaviors and properties that remains currently lacking.

State of the art

Specifying behavior (i.e. dynamic semantics) and formal properties have been a topic of research for quite some time now, resulting with a wide diversity of approaches. In the following we give a brief overview.

Translational semantics are specified through exogenous transformations (i.e. transformations between models expressed using different languages (Mens and Van Gorp, 2006)). The target language should be formally well defined (i.e. it should contain dynamic semantics). Elements from the source domain are translated to elements of the target domain. For instance, the semantics of some UML diagrams are formalized by using translational semantics as proposed in (Clark et al., 2001). The notion of Semantic Anchoring proposed in (Chen et al., 2005), is transformational specification of semantics (i.e. kind of translational semantics), between concepts of DSMLs and selected semantic units (models of computations with associated operational semantics, specified by using Abstract State Machine – ASM). The ASM formal method can be used as an action language to operationally define dynamic semantics as proposed in (Gargantini et al., 2009).

Operational semantics are defined either by action languages or formal behavioral languages. Action languages define operational semantics in ad-hoc manner, as a set of operations associated to each concept of a DSML. This is done by using Object-Oriented languages (e.g. Java), Aspect-Oriented languages (e.g. Kermeta (Muller et al., 2005)), executable constraint languages (e.g. xOCL (Clark et al., 2008)) or other approaches like the MOF action language (Paige et al. 2006). The EPROVIDE framework (Sadilek & Wachsmuth 2008, 2009) allows the specification of operational semantics for a DSML in ad-hoc manner. It is not related to a single technology allowing language designer the possibility to choose between Java, Prolog, ASM or QVT. Xcore (Clark et al., 2004), an extension of EMOF/Ecore, introduces an action language for specifying operational semantics. Similarly, in (Muller et al., 2005) an extension of EMOF by the aspect-oriented language Kermeta is proposed. The process of specifying the operational semantics for a DSML consists in weaving the operational semantics as a set of operations directly into the abstract syntax. *Formal behavioral languages* such as Statecharts, Petri Nett, or Finite Automata, can also be used for the specification of operational semantics. Instead of operations, in this case the behavior is expressed through behavior models (created by using one of the formal behavioral languages) and associated to the DSML's concepts. Real-Time UML (Douglass, 2003) specify dynamic semantics as behavioral models, where: concepts and relationships are given via the class diagram, behavior for each concept is individually defined via the Statechart diagram and the synchronization and event/data exchange is given via the collaboration sequence diagram. Another example is given in (Scheidgen and Fischer, 2007) proposing UML activities and OCL as a formal behavioral language. In contrary to Real-Time UML, this approach associate behavioral models directly to classes' operations. Similarly, in (Mayerhofer et al., 2013) an extension of MOF, named xMOF is proposed. The behavioral is modeled using the fUML's activity diagrams and associated to the classes' operations.

Comparison. Besides the advantage of the facilities and tools available in the target technical space for simulation, formal verification, animation etc., translational semantics have several drawbacks. Models transformation techniques require first expertise and knowledge in the target semantic domain, in transformation languages and tools e.g. ATL (Atlas Transformation Language). Second, demonstrating the "equivalence" between the original, to be checked, model and the transformed model remain limited, often impossible. Finally, results obtained in the target space should be interpreted back according to the concepts of the source space. In contrast, operational semantics has several advantages. Since the domain space is well-known to designers, the behavior is directly defined on concepts. This allows simulating (to analyze an evolution) and animating (to visualize the evolution during simulation) models, as early as possible with minimum effort improving system quality and reducing time-to-market. Using this kind of semantics is preferable for prototyping in particular for simple behaviors that are expressed through discrete states. However specifying

operational semantics with action languages seems related to programming languages. Indeed, it is preferable to model and to formalize dynamic semantics using formal behavioral languages, but there is still a gap between the technical spaces of behavioral languages and the MBSE (Nastov, 2014). Several solutions for bridging this gap are based on model transformation having the previously stated limitations of translational semantics. Alternative approaches overcome such limitations, integrating directly a behavioral language into an already existing metamodeling language (e.g. Run-Time UML (Douglass, 2003), XMOF (Mayerhofer et al., 2013) or the approach proposed in (Scheidgen and Fischer, 2007)). These approaches allows to execute (even partial) UML models, to test them for correctness as early as possible with very little effort, eliminating the need to manually write source code for the model means, removing consequently developer coding defects and thereby improving system quality and reducing time-to-market. However, integrating a behavioral language with a metamodeling language requires additional efforts and work on challenges that are still an open issue. First, the integration process is a complex task and using a design pattern for the best software engineering practices is highly recommended (Buschman et al., 1996). Second, the formal behavioral language has to be carefully chosen for the purpose of verification. Third, behavioral languages should be able to operate with typed input/output data and complex expressions built using typed data.

Properties modeling languages. Several formal languages for property modeling have been proposed in the literature, for instance Temporal Logics (Linear, Constrained…) (Emerson, 1990), OCL or Alloy (Jackson 2002). OCL is complementary to UML and is used to express properties that cannot be defined using the UML's graphical notations. For this OCL predicates specification is verbose based on object-oriented notation and navigation. Alloy is less expressive due to its simple and generic nature and it is designed for simple semantic specification supporting automated lightweight analysis. More globally, applicability and relevance of formal approaches is demonstrated, for instance, for the control system of the Paris metro line 14. The B language (Abrial & Hoare 2005) was used for the automatic generation of 86,000 lines of code that have not experienced software failures since 1997. Among these crucial advantage and other ones, formal approaches allows exhaustiveness and traceability of proofs. However, first, using such approaches in the MBSE context must overcome several obstacles (Chapurlat 2013). Particularly they remain difficult to use, are often considered as time consuming and require particular set of skills, tolls and proof techniques. Second and unfortunately, DSML generally used in MBSE are not semantically well-defined (as previously discussed) because they lack formal specification and created models can be hardly used as base for formal property specification and proof.

Contribution: An approach for designing V&V suitable DSMLs

Remaining the goals presented before, this approach, illustrated in Figure 1, must guide and assist experts (DSML designers as Models designers and Model users) to define DSMLs that can be verified for well-constructiveness and be used to create models that can be simulated and formally checked. We provide in the next theoretical bases and a tooled approach for designing sufficiently "formal" and "precise" DSMLs. This approach is composed 1) of an object oriented metamodeling language, here considered EMOF, 2) a formal behavioral modeling language that is in the current version of our work an extended version of the Interpreted Sequential machine (Vandermeulen, 1996) named eISM, 3) a property modeling language that is CREI (Chapurlat, 2013) and 4) a language for concrete syntaxes (out of the scope of this paper). The result is presented as a single and integrated meta-language "*xviCore*" (executable verifiable and interoperable core concepts and mechanisms) for creating DSML denoted then "*xviDSML*". An xviDSML includes specification of its syntaxes (abstract and concrete), its behavior (dynamic semantics) and properties. Note that, the proposed framework is not related to here described metalanguages (i.e. EMOF, CREI or eISM). In contrary, any metamodeling object-oriented, formal behavioral modeling or property modeling languages can be used instead. We discussed and justify however, our choice hereafter.

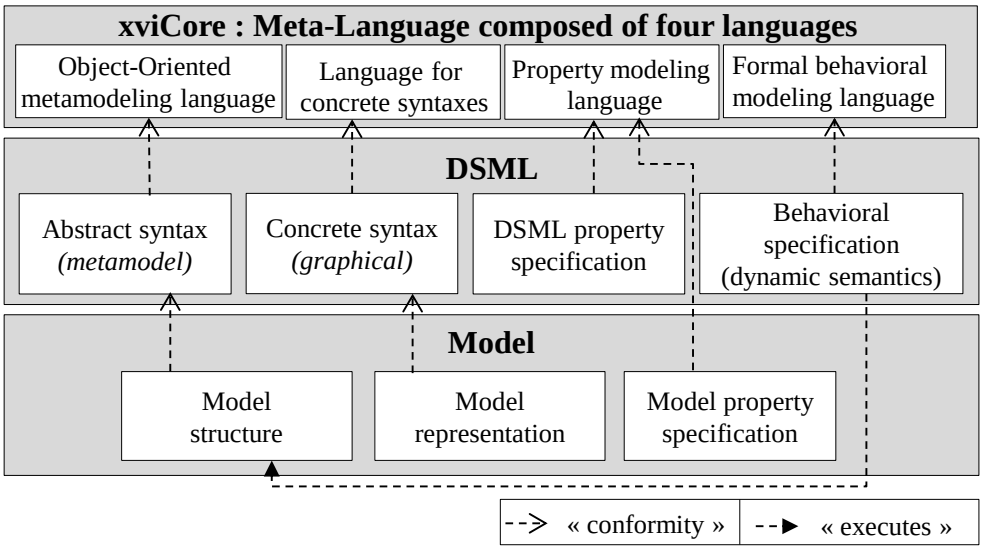

Figure 1. xviCore - a framework for designing DSMLs for V&V objectives.

DSML life cycle and properties modelling

Hereafter followed DSML life cycle is shown in Figure 2 mixing two main phases. The first phase is called "*DSML design time*" during which the DSML syntaxes and semantics are designed following approaches of metamodeling (Pfister et al., 2014) and executable metamodeling (Combemale et al., 2009). The second phase is called "*DSML run time*" which is split up into two sub-phases called respectively "*Model design time*" and "*Model run time*". Indeed, model designer may design then check, simulate and animate or comment then improve if requested one or more models created with the DSML issued form DSML design time. For each phase and sub-phase appear various and specific constraints, expectations and rules to be considered. These ones are modelled as properties having to be checked. We propose structuring properties as synthetized in Figure 2 into *modeling properties* and *system properties*.

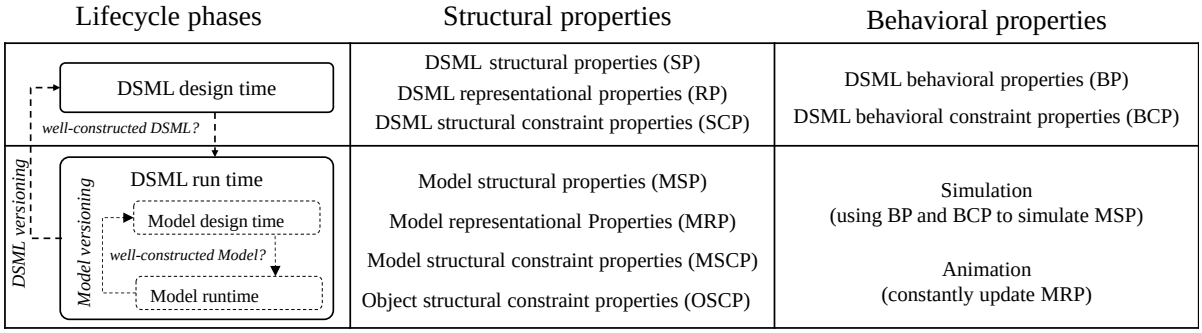

Figure 2: DSML life cycle vs. properties specification / proof and simulation / animation

A modelling property might express features or requirements specific to assume well-constructiveness and coherence of a model or a DSML (DSML or model verification objectives). A system property aims to translate stakeholders' and systems' requirements, functional, physical of operational aspects aiming to assume model relevance thanks to model designer's purpose (DSML or model validation objectives). Modelling and system properties can focus both on static or dynamic (i.e. time dependent) aspects of the DSML or of the pointed out model.

For DSML Design time, modelling properties are structured into:

- DSML Structural Properties (SP): these static properties specify the language's abstract syntax. Metamodeling languages such as EMF/EMOF provide the means for specifying such properties

through a metamodel. For instance eFFBD functional and behavioral modelling language (enhanced Functional Flows Block Diagram) (DoD, 2001) can be designed as a set of classes, each one representing one of the following core elements: *Function*, *Item*, *Resource* and various *Control Constructs* (e.g. parallel, sequential or iterative execution), a set of typed attributes detailing each of these elements (e.g. *quality* and *quantity* of a Resource, *purpose* of a function) and a set of relationships between them (e.g. a relationship *inputs* between *Item* and *Function*).

- DSML Representational Properties (RP): these static properties specify the language representation, forming the language concrete syntax. It includes information about the graphical or textual representation about each of abstract syntax elements. For instance, the graphical representation of the concept Function from the eFFBD language can be defined as a blue rectangular form.

- DSML Structural Constraint Properties (SCP): these both static and dynamic properties are complementary to SP and specify restrictions or supply additional information about the DSML structure that cannot be specified using the SP. They are used to check the well-constructiveness of an abstract syntax, including structural invariants, value derivations, and concepts attributes initial values or types, etc. For instance, the quantity of a provided or a requested resource should always belong to Z+ (always be positive or zero).

- DSML Behavioral Properties (BP): these dynamic properties specify the language behavior, forming the language's dynamic semantics. They are gathered into a behavioral specification that can be implemented using different techniques (action languages e.g. for defining a method "execute" inside Function, behavioral modeling languages as shown in the next, but also transformational languages or executable constraint languages).

- DSML Behavioral Constraint Properties (BCP): these properties complement the BP and specify restrictions or supply additional information about the language behavior. They are used to check the well-constructiveness of a dynamic semantics, including behavioral invariants, pre and post-conditions, etc.

For DSML run time, i.e. Model design time and Model run time, modelling properties are structured into:

- Model Structural Properties (MSP): these static properties specify model's structure in conformity with DSML abstract syntax. So they are based upon SP (abstract syntax) and CSP (structural constraints).

- Model Representational Properties (MRP): these static properties specify model's representation, forming a graphical or textual image inside an editor so they are constrained by the RP.

Finally, last modelling properties and system properties are structured into:

- Model Structural Constraint Properties (MSCP): these both static and dynamic properties are specifically tailored for a given model and do not apply to other models created by using the same DSML. Object Structural Constraint Properties (OSCP): similarly to MSCP, these properties are specifically tailored for a given concept e.g. a Function F1, and do not apply to other instances of the same concept inside a model.

For instance, the property "*a function generally consume more resources than producing in a given time*" can be considered to be satisfied in all eFFBD models as an invariant property (MSCP) but can also be occasionally considered as not satisfied in a particular case in which a function F consume a resource but restore it before releasing (OSCP).

In parallel, models are simulated and animated taking into consideration previously specified properties concerning language's semantics (BP and BCP) but also properties concerning model's semantics (MBCP and OBCP). Model simulation is based on a gradual computation of behavioral rules specified by DSML's dynamic semantics, while model animation is a visual interpretation of a simulation result, according to DSML concrete syntax, i.e., a result to the systematic visualization of changes driven by the computation of the interpretation rules from DSML's dynamic semantics. If errors are detected, the model has to be revised consequently updating and improving it (i.e. model versioning).

Unified Properties Specification Language (UPSL) is proposed to formalize properties. This generic property modelling language has been applied in various fields and particularly in Systems Engineering named "*UPSL-SE*" (Chapurlat, 2013) specifically designed to complement the methodological and technical toolbox for V&V. The goal of this framework is to encourage and ease the work of engineers that are not familiar with formal property modeling languages. USPL-SE is based on the *CREI* (Cause Relation Effect and Indicators) property model. Briefly, a property is a compound entity, composed of causes (*C*) linked to a group of effects (*E*) via a parameterized, temporized and constrained relation (*R*) between *C* and *E*. The relationship formally describes how the set of causes C, when verified, induces a modification on the whole set of effects E and can be eventually evaluated by considering indicators *(I*). CREI language is here used to specify structural and behavioral (i.e. evolution-dependent) properties, characterizing both model and modeled system. For this, USPL-SE takes into consideration the syntaxes and the behavior (dynamic semantics) of a DSML. Last, UPSL-SE authorizes the use of several proof tools such as for instance UPPAAL and a property modeling and proof framework based on Conceptual Graphs as proposed in (Mallek et al., 2012). Due to lack of space, for more details concerning the formalization of UPSL-SE and illustration cases, readers are encouraged to see (Chapurlat, 2014).

Formal Behavioral Modeling Language

The xviCore metalanguage includes a framework for behavioral specification based on the *blackboard design pattern* that is used in the field of Model Driven Engineering (MDE). This pattern is a behavioral pattern "*affecting when and how processes react and perform*". It establishes relationships between independent modules called "*autonomous processes*" that read and write relevant data from/in a "*blackboard*" considered as a shared and structured memory. They are synchronized by a "*controller*" which monitors the properties on the black board and prioritizes autonomous processes (Lalanda, 1997).

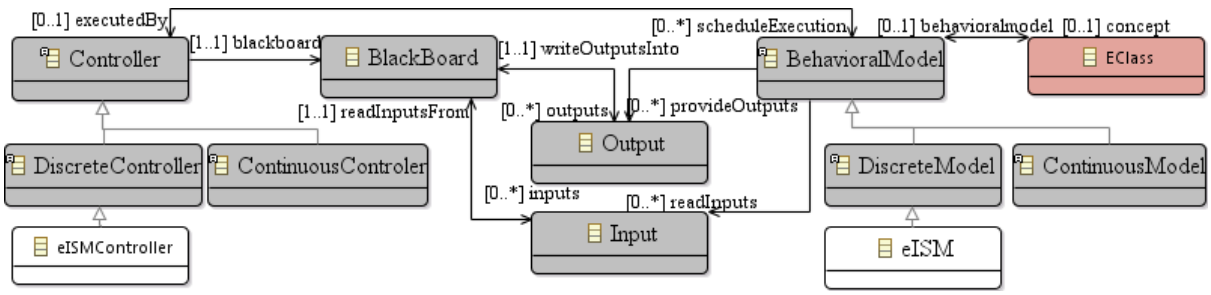

Figure 3. Modeling xviCore following the blackboard design pattern principles.

This framework, illustrated in Figure 3 is then based on:

1) *Controller (C):* modeled by the *Controller* class, it is used to schedule the execution of all behavioral models from a DSML according 1) to a logical clock managing multiscale time and stability hypotheses, and 2) eISM execution algorithm discussed in (Nastov et al., 2014).

2) *Black Board (BB):* modeled by the *BlackBoard* class, BB is the common and time dependent base of information where behavioral models of a DSML write their output data (*O*) and read their input

data (*I*), enabling information exchange. It is formally defined as a 5-uplet $BB \stackrel{\text{def}}{=} \langle AT, LT, V, S, R \rangle$ where: *AT* is the set of variables specifying the time of adding. *LT* is the set of "lifetime" variables, indicating the remaining time before updating messages from the *BT*. *V* is the set of "variables carried out by the messages. *S* is the set of "*sender*" variables specifying the behavioral model that sent the message and $R = \{R_1, .., R_k\}$ is the set of "*receivers*" variables indicating the behavioral models that read the message.

3) *Behavioral model:* modeled by the class *BehavioralModel*, it is used to model the behavior of different DSML concepts that are modeled through classes in the DSML metamodel. It corresponds to the "*autonomous processes*" module of the blackboard design pattern. Behavioral models are designed using a behavioral modeling language that can be based on continuous or discrete events hypotheses. In this article, for discrete behavioral description, we choose eISM. In comparison to other discrete events modeling languages, eISM offers several features. First, it operates with typed input/output data (primitive types, e.g., Boolean, Integer, Real, Character or compound type) and complex expressions built using internal typed data. Second, it separates the states / transitions description from the data specification, limiting the combinatorial explosion of the number of states (i.e., some states can be specified as variables). Finally, models created by using eISM have formal underlying structure, based on the Linear Temporal Logic (LTL) allowing formal verification without transforming an eISM model (Larnac et al., 1997). Last, and in contrast, within the field of MDE, the overall behavior of a DSML is captured through the behaviors of concepts, taken separately, but interacting also with the behavior of other concepts from the same DSML improving then internal interoperability as suggested. Therefore, eISM is able to model such interactions between behavioral models (i.e., behavioral coordination) thanks to the blackboard design pattern, as discussed above. For more details on eISM, please refer to (Nastov et al., 2016).

Application case

We examine here and enrich two well-known languages of SE community: xvi-eFFBD and xvi-PBD. These languages can be then used to create eFFBD and PBD models, simulated and checked following behavioral and properties modelling principles described in previous section. The example consists here to define primary functional and physical architectures of a system S by using respectively xvi-eFFBD and xvi-PBD DSML. S is a liquid transfer system from an external source tanker to an external destination tanker, controlled by a user.

Phase 1: *DSML design time for xvi-eFFBD and xvi-PBD*

eFFBD (Enhanced Functional Flow Block Diagram) is a functional-modeling language allowing system designers to describe both functional and behavioral aspects of complex, distributed, hierarchical, concurrent and communicating systems (Aizier et al., 2012). In (Haskins et al., 2011) a short history of FFBD is described as well as its main evolution eFFBD. The core elements of the eFFBD language evocated in previous section are modelled in Figure 4 that describes partial abstract syntax of eFFBD. In this metamodel, we distinguish a Resource Flow describing requested Resources of a function that consumes them and restores them after execution and an Item Flow describing items that are needed and consumed as inputs for a function or provided as outputs after transformation. eFFBD has several constructs illustrated in Figure 5: Diagram, Branch, Sequence, AND, OR, Iteration and Loop. A Diagram is where the other elements are placed, composed of begin and end operators, a main branch and a set of input/output objects (i.e., items or resources) carried by flows. The branch is composed of various control constructs allowing engineer to describe how functions are chained, put in parallelism, in exclusion... A construct can either be 1) a function control construct composed of a set of functions (eventually one unique function) put in sequence, or 2) an operator control construction containing minimum one branch beginning on a begin operator and ending on an end operator. Four types of operator control construction are introduced: AND, OR, Iteration and Loop.

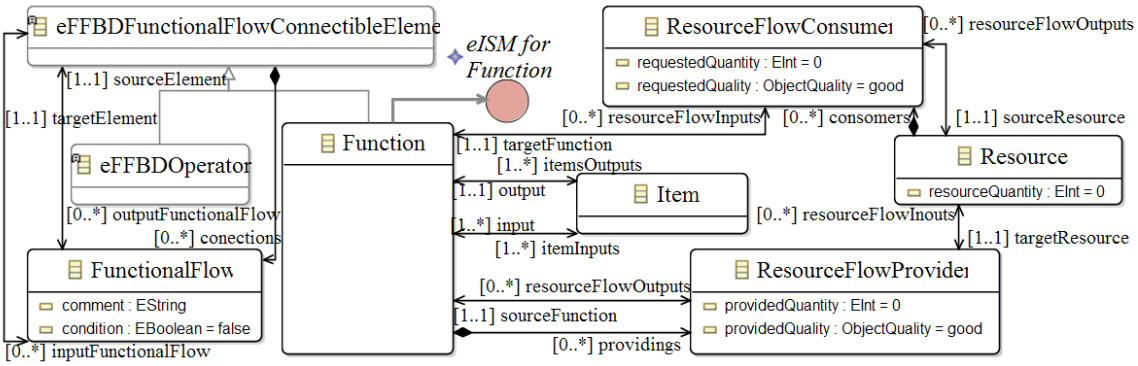

Figure 4. The core elements of the eFFBD language and the flows relating them to each other.

Finally, both packages, illustrated in Figure 4 and Figure 5 are merged into a unique eFFBD metamodel. The next step consists in modeling the eFFBD behavior. For this purpose eISM behavioral models have to be associated to the following concepts: Constructs (i.e. diagram, branch, sequence, iteration, etc.), Function, Item and Resource. Unfortunately, due to lack of space we model hereafter only the behavioral model of Function.

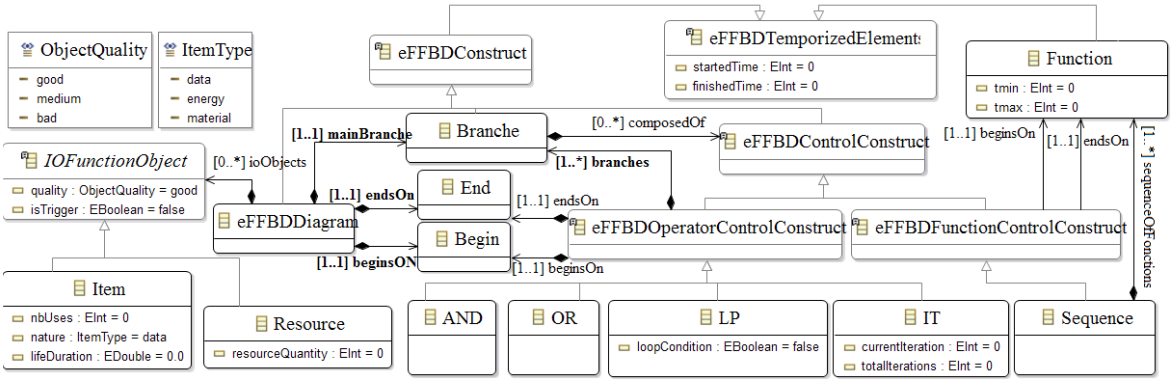

Figure 5. eFFBD modelling language construct (summary)

As illustrated in Figure 6, it is composed of six states: *Sleep*, *Authorized*, *Execution*, *Finished*, *Suspended* and *Aborted*. A Function is initially in the Sleep state, waiting for a request to start execution (*start* event). When the request arrives, the Function enter *Authorized* state, meaning that that input/output transformation is possible depending on the availability of all input Items and Resources as well as the state of the Components on which the Function is allocated (*condition : c1*). When the previous condition is satisfied, the update *transformingInputs* is activated (i.e. the real transformation of energy, material and / or data happens) and the Functions enters *Execution* state. The transformation least a certain time period (*condition: c2*), before producing outputs (*update: providingOutputs*) forcing the Function into *Finished* state. In case of dysfunction of the component (discussed hereafter) on which the function has been allocated (*suspended* event), a function is *Suspended* and eventually *Aborted*, assuming the component does not reply on time (*condition: c5*).

PBD (Physical Block Diagram) is a block-modeling language providing systems engineers with a block-and-line diagram representing the various components and sub-systems and physical links that connect components within a system or system segment (Long, 2007). It offers a more detailed view of the architectural composition.

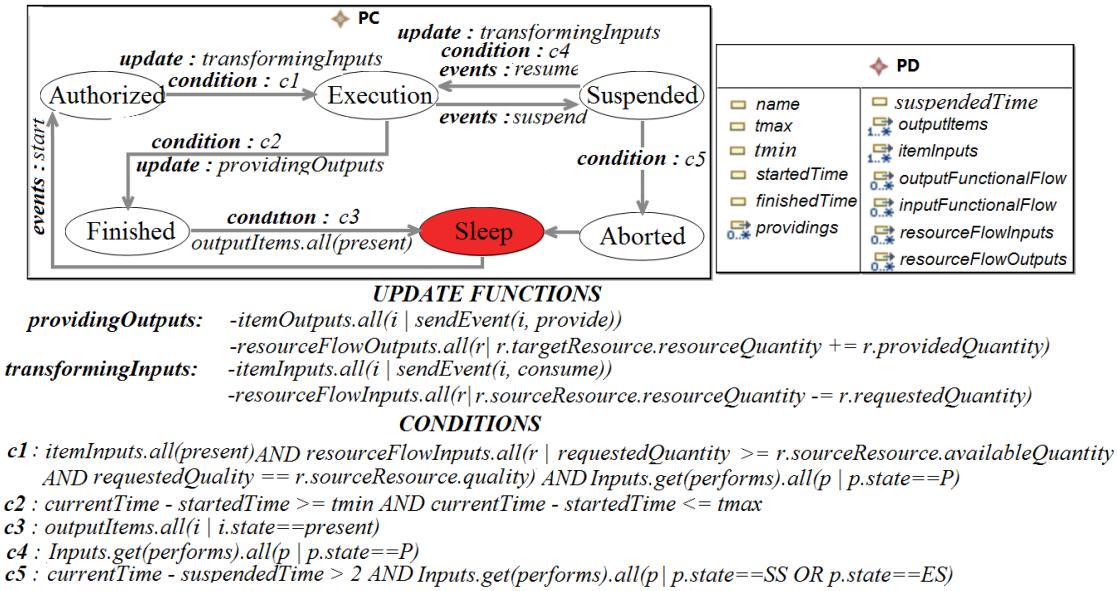

Figure 6. An example of a state model of the Function concept from eFFBD.

The core elements of the PBD language, illustrated in the metamodel of Figure 7, are: Component, Link, Interface and Context.

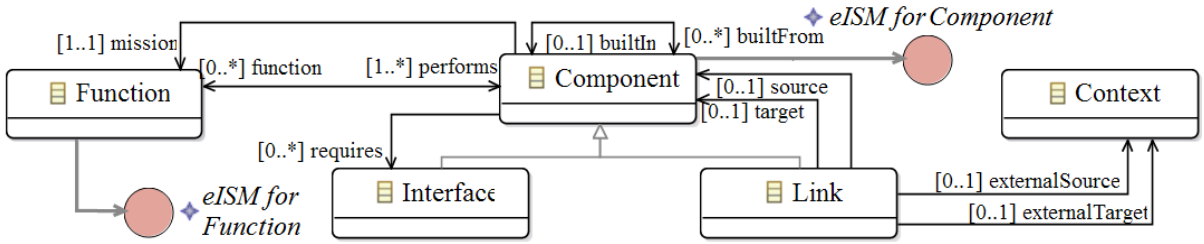

Figure 7. A metamodel of the PBD language (right) and its interactions with the eFFBD language (left).

A Component models a physical part linked by a Link to another component by using Interface. Let's us note that the concept Function from the eFFBD language is linked to Component by a relation allocated to/performs allowing then to describe which functions are allocated then performed by which components. In the same way, a relation allows us to describe how the input, output, and triggers flows of allocated functions are then themselves allocated to a Link devoted then to carry out these flows from external source (Context) or from an existing Component. Once the metamodel is completed, behavioral models should be modeled and associated to the following concepts: *Component*, *Interface* and *Link*. We focus hereafter only on the eISM behavioral model associated to the concept Component as illustrated in Figure 8.

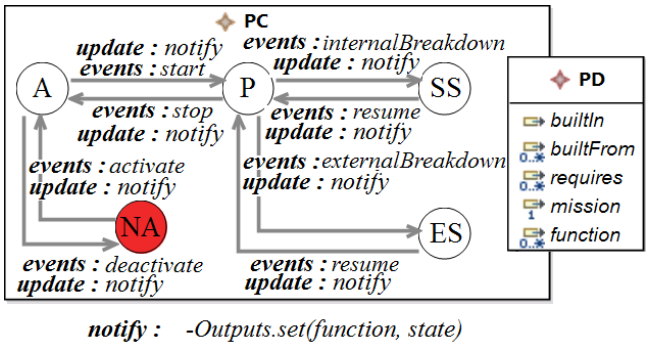

Figure 8. An example of a state model of the Component concept from PBD.

This is composed of five states: non-Active (*NA*), active (*A*), producing (*P*) and breakdown (*SS*) or (*ES*). A component is initially non-active (*NA*) waiting for energy (*activate* event) to get prepared for

a state. When the signal is received, the update *activating* is activated and the component enters activates state (*A*). It starts producing, when the *start* signal is received, activating the update *producing* (i.e. the component performs its function) and it enters producing state (*P*). Components perform their functions until they receive, either a *stop* signal, which put them in the previous state (update *stopping* is activated), or a *breakdown* signal (update *emergency* is activated), which immediately makes them stop producing and puts them in waiting states (*SS* or *ES*) depending on the signal nature (*internal default* or *external default*). Additionally, a component provides its performing functions with its current state (see the *notify* update), allowing them to take the component's current state into account inside their behavioral model (see Function's conditions).

A DSML should also explicate a set of properties concerning its structure or behavior. We previously classified DSML structural constraint properties (SCP) and DSML behavioral constraint properties (BCP). For instance, the SCP "quantity of a provided or a requested resource should belong to Z+ (always be positive or zero)" can be specified as:

$$SCP_1 := \forall f \in \text{Function} \mid f.\text{resourceFlowInputs.forAll}(rfc : ResourceFlowConsumer \mid rfc.\text{requestedQuantity} >= 0)$$

$$SCP_2 := \forall f \in \text{Function} \mid f.\text{providings.forAll}(rfp : ResourceFlowProvider \mid rfp.\text{providedQuantity} >= 0)$$

The used formalism to define BCP depends on the used technique for behavioral specification. In this case, UPSL properties are then transformed into LTL (Linear Temporal Logic) that can be used to check properties for any eISM (Larnac et al. 1997). For instance, the following property: "*there is one and only one current active state unique at any time step*" is defined as (□P means P is always true and oP means P will be true at next moment of the computed evolution of the model) is defined as:

$$BCP_1 := \forall x, y \in \text{States} \mid x \neq y \implies \square(x \supset \neg y)$$

Phase 2: *DSML Run time for xviModels design time and xviModels run time*

xvi-eFFBD and xvi-PBD DSML allows us to define preliminary functional and physical architectures of S. For readability reasons, the tool CORE from Vitech Corp (Long, 2007) has been here used to show xvi-eFFBD and xvi-PBD expected concrete syntaxes illustrated in Figure 9. Equipped modelling, checking and simulation / animation framework implementing xvi-eFFBD and xvi-PBD is currently under development.

First, S behavior can be simulated by linking the two architectural models shown in Figure 9. All Functions (Communicate, Control transfer, Survey and Pump) behaviors are simulated according to the eISM model illustrated in Figure 6. All Components (Source and Destination Tankers, Pump, Controller, Connectors and HMI System) behaviors are simulated according to the eISM model illustrated in Figure 8. The *Controller* schedules (i.e. synchronizes) each of the eISM models associated to Function and Component which are executed based on the evolution algorithm proposed in (Nastov et al., 2014) and using Black Board as previously described. After each eISM evolution cycle, MS Properties are updated. This allows model animation, i.e., MRP update the graphical editor based on the updated data from the MSP. Note that, the MSP and MRP must always conform to their respective SP and RP.

Second, properties such as MSCP and OSCP can be defined. For example, we try to demonstrate:

- *The pump stops working either when no more input liquid remains to pump or when ordered*.

This expectation can be described through the following OSCP:

$$OSCP_1 := \text{Pump.getState}() = \text{NA} \implies \text{Pump.getMision}().\text{getInputItems}(\text{"input liquid"}).\text{quantity} = 0$$

$$\text{OR Pump.getMision().getInputTriggers("}order\text{").getValue() = terminated}$$

Note that this property is described using both architectures (eFFBD and PBD), i.e., we test the current state of the component Pump (PBD architecture) and then we navigate to its mission Pump to verify its input liquid quantity and input trigger orders (eFFBD architecture).

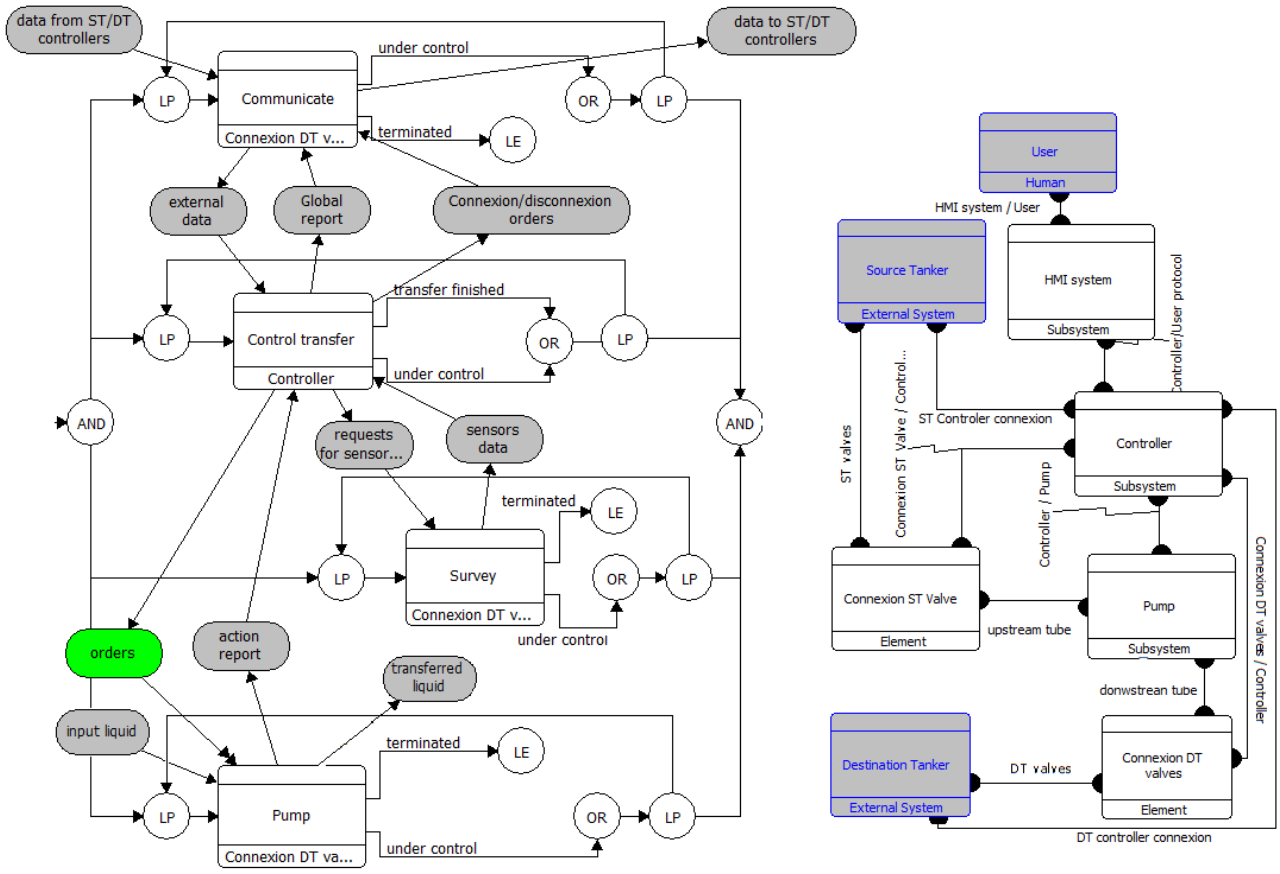

Figure 9. A functional (left) and physical (right) architecture of a liquid transfer system.

- *All functions must stop (i.e. use a Loop Exit construct).* This expectation can be described as an MSCP:

$$\text{MSCP}_1 := \forall \text{lp LP} \mid \text{lp.hasLoopExit()} = \text{true}$$

- *All functions must stop simultaneously under the control of Control Transfer Function, is* finally described as a OSCP:

$$\text{OSCP}_2 := \text{Control Transfer.getLoopCond("trnsfert finished")} = \text{true} \Rightarrow$$
$$\forall \text{f} \in \text{Function, t} <> \text{Control Transfer} \mid \text{t.getLoopCond("terminated")} = \text{true}$$

Conclusion

This paper introduces a model-based methodology, in the form of a tooled approach, for modeling sufficiently formal DSML achieving V&V objectives. It is composed of a metamodeling language (i.e. EMOF), a formal behavioral modeling language (i.e. an extended version of the ISM named eISM) and a property modeling language (i.e. CREI). The result is presented as a single and integrated meta-language "xviCore" (executable verifiable and interoperable core concepts and mechanisms) for creating DSML denoted then "xviDSML". Such xviDSMLs provide the means for designing simultaneously abstract and concrete syntaxes (i.e., domain concepts, relationships and their representations), behavioral specification (i.e., domain concepts behavior) and properties specification (i.e., properties respected by the domain concepts). As a consequence, models created

by such xviDSML can be simulated, animated and checked by using formal proof techniques. In addition, the proposed approach is extensible and therefore not related to introduced metalanguages (i.e. EMOF, CREI or eISM). In contrary, any metamodeling object-oriented, behavioral modeling or property modeling languages can be used instead. Other contributions remain still a subject of a debate. At a final stage, we aim at integrating continuous and hybrid behavior models together with the here presented discrete-events behavioral models.

References

Abrial, J. R., and Hoare, A. 2005. *The B-book: assigning programs to meanings*. Cambridge University Press.

Aizier, B., Chapurlat, V., Lisy-Destrez, S., Prun, D., Seidner, C., and Wippler, J.-L., 2012. "xFFBD: towards a formal yet functional modeling language for system designers," In 22nd Annual INCOSE Symposium.

Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., and Schnoebelen, P. 2013. *Systems and software verification: model-checking techniques and tools*. Springer Science & Business Media.

Buschman, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M., 1996. *A system of patterns: pattern-oriented software architecture*. Wiley & Sons.

Chapurlat, V., 2013. "UPSL-SE: A model verification framework for Systems Engineering." *Computers in Industry*, 64(5), 581-597.

Chapurlat V., 2014. "Property concept and modelling language for Model-Based Systems Engineering (MBSE) context." *Internal Research Report* (open request)

Chen, K., Sztipanovits, J., Abdelwalhed, S., and Jackson, E., 2005. "Semantic anchoring with model transformations." In the European Conference on Modelling Foundations and Applications, ECMFA, pages 115–129.

Clark, T., Evans, E., and Kent, S., 2001. "The Metamodelling Language Calculus: Foundation Semantics for UML. In Conference on Fundamental Approaches to Software Engineering." FASE, pages 17–31, Springer.

Clark, T., Evans, A., Sammut, P., and Willans, J. 2004. "An eXecutable metamodelling facility for domain specific language design."

Clark, T., Sammut, P., and Willans, J. 2008. "Superlanguages: developing languages and applications with XMF."

Combemale, B. Crégut, X. Garoche, P.-L. and Thirioux, X. 2009. "Essay on Semantics Definition in MDE." *An Instrumented Approach for Model Verification. Journal of Software*, 4(6).

de Weck, O. L., Ross, A. M., and Rhodes, D. H. 2012. "Investigating relationships and semantic sets amongst system lifecycle properties (Ilities)." In Third international engineering systems symposium CESUN.

DoD, 2001. "Systems Engineering Fundamentals," Def. Acquis. Univ. Press (available at: http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-885j-aircraft-systems-engineering-fall-2005/readings/sefguide_01_01.pdf last visited 2015-11-02)

Douglass, B. P. 2003. "Real time UML." Formal Techniques in Real-Time and Fault-Tolerant Systems: 7th International Symposium, FTRTFT 2002. Co-sponsored by IFIP WG 2.2. Oldenburg, Germany. Proceedings. Vol. 2469. Springer.

Emerson, E., 1990. "Temporal and modal logic," *Handbook of Theoretical Computer Science*, vol. B. MIT Press. Editor: J. van Leeuwen ISBN 0262220393, pp. 955-1072, 1990

Estefan, J. A. 2008. "Survey of Model-Based Systems Engineering (MBSE) Methodologies 2. Differentiating Methodologies from Processes, Methods, and Lifecycle Models." Jet Propuls., vol. 25, pp. 1–70.

Gargantini, A., Riccobene, E., and Scandurra, P. 2009. "A semantic framework for metamodel-based languages." In Automated Software Engineering, ASE, 16(3-4), 415-454.

Harel, D., and Rumpe, B. 2004. "Meaningful modeling: what's the semantics of "semantics"?" *Computer*, 37(10), 64-72.

Haskins, C., Forsberg, K., and Krueger, M., 2011. "Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities," *Systems Engineering*, INCOSE.

ISO/IEC. 2008. "ISO/IEC 15288 : Systems and software engineering - System life cycle processes." vol. 2008, no. 1. IEEE, p. 5.

Jackson, D. 2002. "Alloy: a lightweight object modelling notation." *ACM Transactions on Software Engineering and Methodology* (TOSEM), 11(2), 256-290.

Kohavi, Z., 1978. “Switching and Finite Automata Theory,” *Computer Science Series,* Tata McGraw Hill.

Kleppe, A. G. 2007. “A language description is more than a metamodel.”

Larnac, M., Chapurlat, V., Magnier, J., and Chenot, B., 1997. “Formal Representation and Proof of the Interpreted Sequential Machine Model.” EUROCAST, Las Palmas.

Long, J. E., 2007 “MBSE in Practice: Developing Systems with CORE,” Vitech briefing slides, Vitech Corporation, Vienna, VA, 2007.

Mahfouz, A. A., Mohammed, M. K., and Salem, F. A. 2013. “Modeling, Simulation and Dynamics Analysis Issues of Electric Motor, for Mechatronics Applications, Using Different Approaches and Verification by MATLAB/Simulink.” *International Journal of Intelligent Systems and Applications (IJISA),* 5(5), 39.

Mallek, S., Daclin, N., and Chapurlat, V., 2012. “The application of interoperability requirement specification and verification to collaborative processes in industry,” *Computers in Industry*, n°63 (2012) 643–658.

Mayerhofer, T. Langer, P. Wimmer, M. and Kappel, G. 2013. “xMOF: Executable DSMLs based on fUML.” In Software Language Engineering (pp. 56-75). Springer International Publishing.

Mens, T., and Van Gorp, P. 2006. “A taxonomy of model transformation.” Electronic Notes in Theoretical Computer Science, 152, 125-142.

Muller, P. A., Fleurey, F., and Jézéquel, J. M. 2005. “Weaving executability into object-oriented meta-languages.” In Model Driven Engineering Languages and Systems (pp. 264-278). Springer Berlin Heidelberg.

Nastov, B. 2014 “Contribution to Model Verification: Operational Semantics for Systems Engineering Modeling Languages,” CIEL, Paris, France.

Nastov, B., Chapurlat, V., Dony, C., and Pfister, F., 2015. “A Verification Approach from MDE Applied to Model Based Systems Engineering: xeFFBD Dynamic Semantics.” In Complex Systems Design & Management (pp. 225-238). Springer International Publishing.

Nastov, B., Chapurlat, V., Dony, C., and Pfister, F., 2016. “Towards Semantical DSMLs for Complex or Cyber-Physical Systems.” In *11th* International conference on Evaluation of Novel Approaches to Software Engineering. Springer International Publishing.

Paige, R. F., Kolovos, D. S., and Polack, F. A. 2006. “An action semantics for MOF 2.0.” In Proceedings of ACM symposium on Applied computing (pp. 1304-1305). ACM.

Pfister, F., Huchard, M., and Nebut, C., 2014. “A Framework for Concurrent Design of Metamodels and Diagrams - Towards an Agile Method for the Synthesis of Domain Specific Graphical Modeling Languages.” In the International Conference on Enterprise Information Systems, Lisbon, Portugal.

Rivera, J. E., Romero, J. R., and Vallecillo, A. 2009. “Behavior, time and viewpoint consistency: Three challenges for MDE.” In Models in Software Engineering (pp. 60-65). Springer Berlin Heidelberg.

Rozier, K. Y., 2011. “Linear temporal logic symbolic model checking.” *Computer Science Review*, 5(2), 163-203.

Sadilek, D. A., and Wachsmuth, G. 2008. “Prototyping visual interpreters and debuggers for domain-specific modelling languages.” In Model Driven Architecture–Foundations and Applications (pp. 63-78). Springer Berlin Heidelberg.

Sadilek, D. A., and Wachsmuth, G. 2009. “Using grammarware languages to define operational semantics of modelled languages.” In Objects, Components, Models and Patterns (pp. 348-356). Springer Berlin Heidelberg.

Scheidgen. M, and Fischer, J. 2007. “Human comprehensible and machine processable specifications of operational semantics.” In the European Conference on Modelling Foundations and Applications, ECMFA, volume 4530 of LNCS - pages 157–171. Springer.

Vandermeulen, E., 1996. “Machine Séquentielle Interprétée.” PhD Thesis University of Montpellier II (in French).