A Survey of Model-Driven Approaches Applied to DEVS
A Comparative Study of Metamodels and Transformations

Stéphane Garredu, Evelyne Vittori, Jean-François Santucci and Bastien Poggi
Department of Computer Science, University of Corsica, Campus Grimaldi, Corte, France

Keywords: M&S, DEVS, MDE, MDA, M2M, M2T, Interoperability.

Abstract: Model-Driven Engineering (MDE) is a subset of Software Engineering (SE) which focuses on models. MDE provides techniques and guidelines to create models (metamodeling) and to transform them onto other models (including code). Recently, several MDE approaches have been successfully applied to the world of Modeling and Simulation (M&S), of which DEVS (Discrete EVent system Specification) is one of the most popular formalisms. The goal of those approaches is to increase DEVS interoperability. Many of them resort to a metamodel to describe DEVS concepts. The most recent ones also provide automatic code generation "Model-To-Text" (M2T) towards DEVS simulators (DEVS "internal" interoperability) and establish links between DEVS and other formalisms, thanks to Model-To-Model (M2M) transformations (DEVS "external" interoperability). The purpose of this paper is to give a state of the art of the MDE contributions to DEVS formalism and to provide a comparative study of the most recent ones.

1 INTRODUCTION

Model-Driven Engineering (MDE) is a set of methods, approaches and techniques inherited from Software Engineering (SE). The common point shared by all of the MDE-oriented approaches is the use of models.

DEVS (Discrete EVent system Specification) (Zeigler 1976) formalism relies on a strong mathematical background, inspired by the set theory, and enables to create models, which can be interconnected, and to simulate them. To simulate a DEVS model, it is needed to make a move from a theoretical model into a concrete implementation.

There exist several different DEVS-oriented frameworks, lying on different object-oriented languages, used by several research teams in the world: that induces a lack of interoperability between DEVS models, which cannot be reused by the same team on another DEVS-oriented platform, and even less be shared among the whole DEVS community. But this lack of interoperability logically generates a need too. This need for interoperability between DEVS implemented models gave rise to several approaches, and a significant part of them is inspired by MDE.

This paper is dedicated to those MDE contributions to DEVS formalism. We chose to highlight the approaches which propose a meta-model for DEVS and involve transformation mechanisms. We present a comparative study of those approaches, focusing on three key aspects: the way they handle the DEVS basic concepts, the underlying meta-formalism, and the motivations of the work (improving interoperability, code generation…).

This paper is organized as follows: it starts with a background section, dedicated to the DEVS formalism, and the key elements of MDE. The following section presents some of the MDE approaches that have been applied to DEVS formalism, and compare them. Finally, we conclude with a short discussion on the actual and future challenges in DEVS interoperability using MDE.

2 BACKGROUND

2.1 DEVS Formalism

Since the 1970s, formal approaches have been proposed for the modeling and the simulation discrete event dynamic systems, and the DEVS formalism is a part of them. This formalism may be defined as a universal and general methodology,

Garredu S., Vittori E., Santucci J. and Poggi B..
A Survey of Model-Driven Approaches Applied to DEVS - A Comparative Study of Metamodels and Transformations.
DOI: 10.5220/0005041001790187
In *Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH-2014), pages 179-187
ISBN: 978-989-758-038-3

which provides tools to model and simulate systems, whose behavior is based on events. This formalism lies on the system theory and permits the specification of complex discrete event systems in a modular and hierarchical way.

DEVS has been implemented on several platforms, for instance DEVSJAVA (DEVSJAVA 2013) or PyDEVS (Bolduc et al. 2001).

At this time, there exist no unique (standardized and platform-independent) representation of DEVS models. Usually, a simple DEVS model, with a one-dimensional (qualitative) state variable, is graphically represented as a kind of annotated finite automaton. If it needs to be more accurate, its tuples are described mathematically and, when necessary, a pseudo-language is used. Nevertheless, most of the time, a DEVS model only exists under its implemented form (object-oriented code). Several approaches have been trying to propose a standard representation of DEVS models: we can quote here the work of the SISO (Simulation Interoperability Standards Organization) (SISO 2008). DEVS is composed of two artifacts: the atomic models and the coupled models.

2.1.1 DEVS Atomic Model

The basic entity in DEVS is the atomic model:

$AM = < X, Y, S, ta, \delta_{int}, \delta_{ext}, \lambda >$, where :

- $X = \{(p,v) | p \in InputPorts, v \in X_p\}$ is the input events set; *InputPorts* is the set of input ports and X_p is the set of possible values for those input ports;
- $Y = \{(p,v) | p \in OutputPorts, v \in Y_p\}$ is the output events set; *OutputPorts* is the set of output ports and Y_p is the set of possible values for those output ports;
- S is the states set of the system;
- $ta: S \rightarrow R_0^+ \cup +\infty$ is the time advance function (or lifespan of a state);
- $\delta_{int}: S \rightarrow S$ is the internal transition function;
- $\delta_{ext}: Q \times X \rightarrow S$ with $Q = \{(s,e) / s \in S, e \in [0, ta(s)]\}$ is the external transition function;
- $\lambda: S \rightarrow Y$, with $Y = \{(p,v) | p \in OutputPorts, v \in Y_p\}$ is the output function.

2.1.2 DEVS Coupled Model

The purpose of a DEVS coupled model is to describe a hierarchy: it has sub-models (which can be either atomic or coupled) and couplings between them. A coupled model is formally defined by:

$CM = <X, Y, D, \{Md | d \in D\}, EIC, EOC, IC, select>$

Where

- *X* and Y are the same as in 2.2.1)
- *D* is the set of component names, $d \in D$;
- M_d is a DEVS model (atomic or coupled);
- *EIC* is the set of external input couplings; an external input coupling is a link between the input port of the current coupled model and the input port of any of its sub-models;
- *EOC* is the set of external output couplings; an external output coupling is a link between the output port of the current coupled model and the output port of any of its sub-models;
- *IC* is the set of internal couplings; an internal coupling is a link which involves the output port of a sub-model and the input port of another sub-model;
- *select* is the tiebreaker (selection) function.

2.2 DEVS Interoperability

Throughout this article, we frequently refer to the idea of *interoperability*. We begin with a proposal for a simple classification of the different kinds of DEVS interoperabilities, in terms of range of use: "internal" or "external". Then we give an overview of the two different families of approaches which aim to increase DEVS interoperability: simulator-based or model-based. In this paper, we focus on the second one.

2.2.1 "Internal" vs. "External" Interoperability

Even when it is employed within a DEVS context, the word *interoperability* can be understood differently, depending on the situation.

For instance, let us consider the ability that two DEVS coupled models have to be immediately simulated together. Although they both are DEVS models, it's not possible to simulate them if they are not implemented in the same language (according to the same simulator). Hence, we introduce here the concept of "internal" interoperability, because this problem takes place in a DEVS world, between two DEVS simulators. This is a typical case of an issue which can be solved using a DEVS simulator-based

interoperability approach. The ability for a DEVS model to be exported onto several platforms also refers to an "internal" interoperability because, once again, this problem remains in the DEVS world. This is a typical case where a model-oriented interoperability approach can be successfully applied.

On the other hand, if we consider the ability for a DEVS model to be simulated with a non-DEVS model, it is obvious that this interoperability problem is different from the previous ones. Instead of remaining in a DEVS world, we have to deal with two different formalisms. Here we deal with formalisms and not implementation. We propose to call "external" the interoperability between DEVS and another formalism.

Those internal and external interoperability issues have been explored in (Garredu et al. 2012) and (Garredu et al. 2013).

2.2.2 Simulator-Based vs. Model-Based Solutions

During the last decade, several efforts have been made to fill the gap between the different existing DEVS frameworks. Two major kinds of solutions have been used. An overview of them can be found in (Touraille et al., 2009).

The first one aims to increase the interoperability from the point of view of the simulators, using, for instance, standard messages between at least two different simulation platforms where different models are defined. As an example, we can quote (Seo, 2009) which is a simulator-oriented proposal for a better DEVS simulator-based interoperability using Service-Oriented Architecture.

2.3 Model-Driven Engineering

MDE is a generic software development methodology that focuses on creating, exploiting and transforming models: in fact, everything in MDE is seen as a model, i.e. an abstraction of a part of the real world, described with a modeling formalism. One of the most popular MDE incarnations is the Object Management Group (OMG) Model-Driven Architecture (MDA) (OMG 2001).

2.3.1 Abstraction Levels in MDE

Metamodels and models are linked to each other at different abstraction levels, in the following way: a model (M_1 level) *describes* the real world (M_0 level), and it *conforms to* its metamodel (M_2 level). A metamodel *describes* a modeling formalism, and *conforms* itself to a *metaformalism* (M_3 level). The metaformalism is located at the top of the MDE "meta" levels hierarchy: it contains its own description and conforms to itself: there is no higher level than M_3. The concepts of conformance and description are detailed in (Bézivin, 2004) and shown in Figure 1.

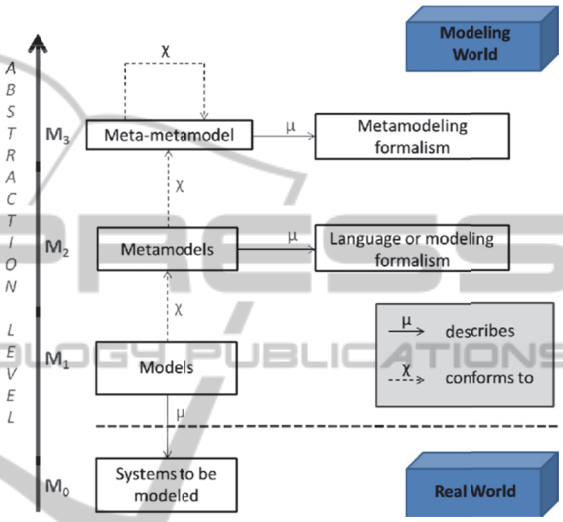

Figure 1: Abstraction levels in MDE.

2.3.2 Model Transformations in MDE

Model transformations are, with models and metamodels, the key concepts of MDE. A transformation is defined at the metamodel level (M_2) and executed at the model level (M_1). The general case for a model transformation is the following one: if MMx and MMy are two distinct metamodels, and Mx is a model which conforms to MMx, the associated transformation is defined between MMx and MMy and executed between Mx and My. It transforms Mx into another model My. Here, we assume that there exists a mapping between the formalisms described by MMx and by MMy, which is not always the case.

Each metamodel, according to Figure 1, conforms to a different metaformalism, and the transformation itself, as a model, conforms to a metaformalism too, which can be, in this case, MFx or MFy. If MMx and MMy are different, the transformation is called *exogenous*. If MMx and MMy are the same metamodel, the transformation is called *endogenous*. Usually, a Model-To-Model (M2M) transformation follows the schema

presented in figure 2. Even if a Model-To-Text (M2T) transformation is, in theory, a model transformation (from a MDE-based point of view), its implementation is often simplified: it directly turns a model Mx into source code, without resorting to a target metamodel. Many existing model transformation techniques are detailed in (Mens et al. 2006).

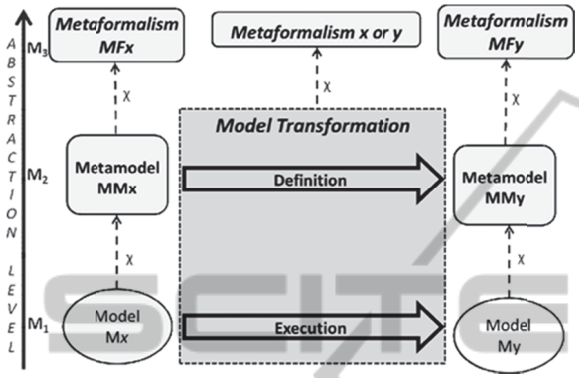

Figure 2: A classic MDE transformation.

2.3.3 A Particular MDE Incarnation: MDA

The MDA approach (OMG 2001) (Kleppe et al. 2004) is very similar to the architecture shown in Figure 2. At the top level lies the metaformalism MOF (Meta-Object Facility). One of the most popular metamodels used in MDA-oriented approaches is the Unified Modeling Language (UML) (OMG UML 2011) (Rumbaugh et al. 2005).

Nevertheless, the adaptability of UML can be a drawback when specific concepts have to be expressed. To solve this, the OMG proposes two solutions: use the specialization mechanisms of UML (profiles), or define a completely new metamodel (conforming to the MOF) which will be used in place of UML.

One of the most famous applications of the former is the OMG SysML (SYStems Modeling Language) (OMG SysML 2012), which can be considered as an extension of a subset of UML. The latter offers a larger set of possibilities, even if it is often more complicated to create a new metamodel instead of using an existing one.

MDA uses three different kinds of models: CIMs (Computation Independent Models), PIMs and PSMs (Platform Independent/Specific Models). Some famous MDA-inspired transformation tools/languages are Acceleo (ACCELEO 2013), dedicated to M2T transformations, and ATL (Jouault et al. 2006) dedicated to M2M. They both are available as Eclipse Modeling Framework (EMF) (Steinberg et al. 2009) plugins.

3 MDE APPROACHES APPLIED TO DEVS FORMALISM

In this section, the ideas and concepts presented in the background section are put together. We present a state of the art of the MDE approaches that have been applied to DEVS, in a model-oriented way. The main purpose of those approaches is to represent DEVS models, focusing on concepts rather than code.

We have previously stated that, when specific domain concepts had to be expressed (i.e. when it is necessary to resort to a metamodel), well-known metamodels could be reused (and specialized) or new ones could be created.

The biggest problem that remains when defining a metamodel for DEVS is not the representation of the structure (coupled models), but the description of the behavior (atomic model), with the DEVS atomic functions. In broad outline, it can be said that the first (until 2007) MDE-like approaches applied to DEVS were based on existing meta-models (see 3.1), while the most recent ones (since 2007) are based on new metamodels (see 3.2). At the moment, these seem to be the most promising ones.

3.1 UML-Like Approaches (before 2007)

Those MDE approaches tackled the lack of interoperability of the DEVS formalism by reusing existing metamodels, or more exactly only one metamodel: UML. Hence, UML is their common denominator. The main advantage of those approaches is the use of UML as a metamodel (abstract syntax), so that the user can use graphical notations as an abstract syntax. Globally, the DEVS models written in UML are well-represented and well-documented. However, an important drawback is that it is often difficult to obtain simulation code from those UML models, except (Risco-Martin et al. 2007) who use tools related to the XML technological space.

One of the first UML-Like approaches applied to DEVS was (Schulz et al. 2000). It resorted to the Statecharts formalism (Harel 1987) to create models that were equal to DEVS models. In the same philosophy, (Risco-Martin et al. 2007) proposed to use XML in order to translate Statecharts formalism (known as UML state

machine diagram) into DEVS. This goes back to create DEVS models using Statecharts.

Some researches proposed to use more than one UML diagram. For instance, (Zinoviev 2005) claims that DEVS suffers from a lack of graphical representation and uses UML to represent DEVS models. Atomic models are specified using UML state machine diagrams, while coupled models are described with UML component diagrams. This approach was not given any concrete implementation. On the other hand, (Mooney et al. 2009) proposed a complete environment that allows the execution of DEVS models specified with UML: this approach tackles the temporal aspects, which are more present in DEVS than they are in UML.

Other approaches rely on the UML extension mechanisms. Some use existing UML profiles, this is the case for (Nikolaidou et al. 2008). The authors employ SysML to graphically represent DEVS models created with DEVSML. Others create special profiles, for instance (Nikolaidou et al. 2008). The authors also provide a partial code generation (skeleton).

3.2 Recent MDE Approaches (since 2007)

Usually, all the MDE approaches we study here follow the same pattern, inherited from MDE philosophy:

- Define a metamodel for DEVS
- Create M2T transformations (and/or)
- Define M2M mappings

The definition of a metamodel for DEVS is the first step, it makes one able to define DEVS atomic and coupled models. To create it, the starting point is the basic definition of DEVS formalism (2.1);

In the second step, links between the DEVS metamodel and DEVS simulators (code) are established. The goal here is to reach several DEVS frameworks from only one DEVS platform-independent model (internal interoperability). Hence, a single model would be simulated on several platforms (M2T);

The third step establishes links between the DEVS metamodel and other formalisms. Those links tackle the external DEVS interoperability problem using M2M techniques.

Regardless of the technique used, the most essential element that directly influences the viability, the power of expression and the accuracy of the DEVS models is the quality of the metamodel they lie on.

The approaches we study in the next subsections take place within different *technological spaces* (Bézivin et al. 2005). The metaformalisms used by the following approaches are: XML Schema, MOF-Ecore, EBNF (Extended Backus-Naur Form), E/R Diagrams.

3.2.1 XML-Based Approaches

DEVSML (Mittal et al. 2007) is a metamodel used for the description of DEVS models within Service-Oriented Architectures (SOA). This approach enables to specify DEVS models but it is not totally platform-indepndent: the models can only be used with Java-based DEVS platforms, because DEVSML inherits from a XML-like language used to describe Java programs, JavaML (Badros 2000).

More recently, (Mittal et al. 2012) introduced the "DEVSML framework" where it is possible to create Domain-Specific Languages (DSLs) and associate them to the DEVSML language. It is also possible to generate DEVSJAVA (DEVSJAVA 2013) simulation code. This MDE-oriented approach is implemented within the Eclipse Modeling Framework (EMF) but uses the EBNF metaformalism.

There also exists a similar approach: SimStudio (Touraille et al. 2011) based on a metamodel named DEVS Markup Language (DML). The idea is to improve DEVS internal interoperability using "hybrid code": some same parts of the code are written in different languages so they can be chosen during the code generation.

DEVSML has been used for M2M transformations towards DEVS (Risco-Martin et al. 2007) (see 3.1). The M2T implementations use the language Xtend while all the transformations in SimStudio use XSLT (eXtensible Stylesheet Language Transformations).

3.2.2 E/R Diagrams-Based Approaches

The metamodeling environment AToM^3 (Lara et al. 2002) has been used by (Posse et al. 2003) and after that by (Levytskyy et al. 2003) to create a DEVS metamodel based on the E/R Diagrams metaformalism. Those approaches use the abilities of AToM^3 in order to generate a graphical modeling environment based on the DEVS meta-model. All the DEVS states are enumerated (sequential states).

Another DEVS metamodel, introduced in (Song 2006), allows specifying state variables as attributes. The transition functions are specified with text blocks. For all of those approaches, it is

possible to generate Python code for the simulator PyDEVS (Bolduc et al. 2001) using M2T mechanisms. Moreover, some M2M approaches were implemented, for instance (Borland 2003) proposed a transformation from Statecharts to DEVS. Code generation and model transformations are performed using the Python language.

3.2.3 MOF-Based Approaches

Those approaches are the most recent ones. They are located in the object-oriented technological space, and use MOF (and its subset EMF Ecore) as a metaformalism: in other words, they can be considered as MDA approaches.

(Lei et al. 2009) use a M2M transformation from DEVS to SMP2 (Simulation Model Portability 2). To do so, they resort to two packages. One of them contains the definition of the DEVS formalism. The DEVS functions are taken into account but the programmer needs to fill them: otherwise, they remain empty. The states are explicitly enumerated, under a textual form.

On the contrary, EMF-DEVS (Sarjoughian et al. 2012) is only centered on the DEVS formalism and its internal interoperability (EMF-DEVS does not provide any M2M transformation). The Ecore metamodel proposed by this approach enables to generate Java code using the native M2T EMF mechanisms. The atomic functions are abstract, and must be implemented manually. Only the coupling functions are automatically generated.

MDD4MS (Cetinkaya et al. 2012) was originally based on the GME environment, but it is now fully implemented within EMF Ecore. The authors follow a MDA-oriented approach to define a DEVS metamodel. The atomic functions are specified using a platform-independent pseudo-code, which seems to be described itself by a metamodel, linked to the DEVS metamodel. The states are handled by state variables, they can be typed and they also can be affected an initial value. A total code generation is provided, towards Java platforms (M2T). A M2M transformation from BPMN to DEVS is also proposed.

MetaDEVS (Garredu et al. 2012) is a metamodel for DEVS based on a MDA approach, implemented within the Eclipse EMF framework. It specifies the DEVS states using typed variables that can be either enumerated or not. The functions are defined in a platform-independent way, using the DEVSRule concept based on Conditions and Actions. Code generation mechanisms use Acceleo. A generic approach form M2M transformations have also been proposed in (Garredu et al. 2013) and applied to a transformation between FSMs and DEVS.

3.2.4 Comparison of the MDE Approaches

Those approaches can be compared following several criteria. As far as we are concerned, the most significant ones are the solutions chosen to express states and functions. All of them favor finite and enumerated states and some ones also use state variables: (Song 2006), (Mittal et al. 2012), (Touraille et al. 2010), (Lei et al. 2009), (Cetinkaya et al. 2012), (Garredu et al. 2012). The advantage of taking into account the state variables in addition to enumerated states is that multidimensional states can be specified. In our opinion, these solutions appear to be complementary.

Modeling the behavioral functions is also a criteria to evaluate the DEVS metamodels. Three different approaches are used.

The first one is proposed by the approaches that chose to represent the states in a finite way: (Posse et al. 2003), (Song 2006), (Mittal et al. 2012), (Lei et al. 2009). In this case, the transition functions are enumerated and explicitly specify the state changes that must occur. The time advance and output functions are instantiated with each state: therefore, a state contains its lifespan and the associated output. The main advantage of the approach is that it leads to a complete automatic code generation, while its main drawback is that it is not able to specify more complex logical behaviors.

The second one, chosen by (Song 2006), (Sarjoughian et al. 2012) and (Risco-Martín et al. 2007) is depending on the platforms. It defines the elements that must be specified, or completed, by the programmer. Those elements depend on the target platform. Hence, some approaches consider the atomic functions are considered as abstract methods, textual meta-attributes, or code blocks. To provide a complete code generation, the same function must be written in several languages, but the logic they specify is not platform-independent (Touraille et al. 2010).

Finally, the third one uses specifics metamodels in order to allow creating behavioral rules that define all the functions in a platform-independent way. The goal is to perform automatic code generations by limiting as much as possible the intervention of the programmer. The two main solutions aim to use a pseudo language (Cetinkaya et al. 2012) or a behavioral logic within the

Table 1: Comparison of the recent DEVS-MDE approaches.

Approach	Metaformalism	Enum. States	State Vars	DEVS Functions ($\delta_{int} - \delta_{ext} - \lambda - ta$)
AToM^3 DEVS V1 (Posse et al. 2003)	ER Diagrams	YES	NO	Finite
AToM^3 DEVS V2 (Song 2006)	ER Diagrams	YES	YES	Finite or code blocks
DEVSML V2 (Mittal et al. 2012)	EBNF	YES	YES	Finite (DEVS State Machines)
DEVSML V1 (Risco-Martín et al. 2007)	XML Schema	YES	NO	Code Blocks
SimStudio/DML (Touraille et al. 2010)	XML Schema	YES	YES	Partially Described (Hybrid code)
DEVS to SMP2 (Lei et al. 2009)	MOF Ecore	YES	YES	Finite
MDD4MS (Cetinkaya et al. 2012)	MOF Ecore	YES	YES	Platform-Independent Pseudo-Code
EMF-DEVS (Sarjoughian et al. 2012)	MOF Ecore	YES	NO	Abstract Methods
MetaDEVS (Garredu et al. 2012)	MOF Ecore	YES	YES	Platform-Independent Meta-Classes

Table 2: Transformations associated to some recent DEVS-MDE approaches.

	M2M Transformation		M2T Transformation	
Approach	Nature	Transformation Approach	Destination Platform	Transformation Approach
AToM^3 DEVS V1 (Posse et al. 2003) and (Borland 2003)	SC → DEVS	Graph (Python)	PyDEVS	Model Parsing (Python)
DEVSML V2 (Mittal et al. 2012)	DSLs→DEVSML	Xtend	DEVSJAVA	Xtend
DEVSML V1 (Risco-Martín et al. 2007)	SC→DEVS-SM	XML Parser	DEVS-XML	XML Parser
SimStudio/DML (Touraille et al. 2010)	x	x	DML-Lang & DML	XML Parser
DEVS to SMP2 (Lei et al. 2009)	DEVS → SDML	ATL	x	x
MDD4MS (Cetinkaya et al. 2012)	BPMN → DEVS	ATL	DEVSJAVA	Model Parsing (Java)
EMF-DEVS (Sarjoughian et al. 2012)	x	x	DEVS-Suite	Model Parsing (Java)
MetaDEVS (Garredu et al. 2013)	FSM → DEVS	ATL	PyDEVS & Other	Acceleo (template)

associated DEVS metamodel (Garredu et al. 2012). Table 1 summarizes our comparison.

Moreover, some of the approaches were used in M2M and/or M2T contexts. Some approaches are located in the OMG MDA technological space and use M2M transformation languages as ATL: (Lei et al. 2009), (Cetinkaya et al. 2012), (Garredu et al. 2012). Other approaches use different transformation techniques (because of the technological spaces they belong to), such as XML for (Risco-Martín et al. 2007) or Python for (Borland 2003). Some approaches do not perform, at this time, M2M transformations.

The transformations associated to the approaches we have presented are shown in Table 2. Some information about their M2M (DEVS external interoperability) and M2T (DEVS internal interoperability) aspects is provided.

4 CONCLUSION

In this article, an overview of some significant MDE-oriented approaches linked to DEVS was presented. Some of them were detailed and compared. Using a MDE approach in a DEVS context increases the lifetime of the models, improves the way they are defined, makes them be reusable, and enhances their interoperability with other platforms and even other formalisms. Many MDE-oriented tools have been developed, in

particular for the Eclipse EMF platform, as additional plug-ins.

However, the metamodels that have been proposed for DEVS face a difficult issue: the definition of the states and the transition functions in a platform-independent way. Doing so highly reduces the power of expression of the metamodel. Some research need to be done in order to increase the power of expression of the metamodels, maybe with a combination of graphical and textual notations.

An important criteria, which was not evaluated here, is linked to the semantics of the metamodels: indeed, a metamodel only specifies an abstract syntax and needs semantics to be more accurate. MDAbased meta-models usually resort to Object Constraint Language (OCL) to express those semantics. However, the power of a metamodel's semantics remain hard to evaluate.

REFERENCES

ACCELEO2013. ttp://www.eclipse.org/acceleo/.

Badros, G., 2000. "JavaML: A Markup Language for Java Source Code*." Proceedings of the 9th International World Wide Web Conference (Amsterdam, Netherlands, May. 15-19),159- 7.*

Bézivin J., « Sur les principes de base de l'ingénierie des modèles », RSTI-L'Objet, 10(4):145–157, 2004.

Bézivin,J and Kurtev, I. : Model-based Technology Integration with the Technical Space Concept. In Metainformatics Symposium, Esbjerg, Denmark, 2005. Springer-Verlag.

Bolduc, J.S., Vangheluwe, H. A modelling and simulation package for classical hierarchical DEVS. MSDL technical report MSDL-TR-2001-01, McGill University, June 2001

Borland, S., Transforming Statechart models to DEVS,2003.

Cetinkaya D., Verbraeck A., and Seck M. D., Model transformation from BPMN to DEVS in the MDD4MS framework, Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium, Orlando, Floride, 2012

DEVSJAVA2013. http://www.acims.arizona.edu/
SOFTWARE/software.shtml#DEVSJAVA.

Garredu, S., Vittori, E., Santucci, J.-F., and Bisgambiglia, P.-A., A Meta-Model for DEVS - Designed following Model Driven Engineering Specifications, *Proceedings of the 2nd International Conference on Simulation and Modeling Methodologies, Technologies and Applications, Simultech 2012,* Rome, Italy, 28 - 31 July, 2012.

Garredu, S., Vittori, E., Santucci, J.-F., and Bisgambiglia, P.-A., (In Press) From State-Transition Models to DEVS Models - Improving DEVS external interoperability using MetaDEVS: a MDE approach, *Proceedings of the 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications, Reykjavik, Simultech 2013, Iceland, 29 - 31 July, 2013.*

Harel D., Statecharts : A visual formalism for complex systems, Science of Computer Programming, 8(3):231-274, 1987.

Jouault, F. and Kurtev, I. (2006) On the architectural alignment of ATL and QVT. In: Proceedings of the 2006 ACM symposium on Applied computing, Dijon, France.

Kleppe, A., Warmer, S., Bast, W., "MDA Explained. The Model Driven Architecture: Practice and Promise", Addison-Wesley, April 2003.

Lara J., Vangheluwe, H., "Using AToM as a Meta CASE Tool", 4th International Conference on Enterprise Information Systems, Universidad de Castilla-La Mancha, Ciudad Real (Spain), 3-6, April 2002.

Lei, Y., Wang, W., Li, Q., and Zhu, Y., A transformation model from DEVS to SMP2 based on MDA, Simulation Modelling Practice and Theory, Vol. 17, Nr. 10 (2009) , p. 1690-1709.

Levytskyy, A., Kerckhoffs, E. J., Posse, E. and Vangheluwe, H., "Creating DEVS components with the meta-modelling tool AToM^3" in 15th European Simulation Symposium (ESS), A. Verbraeck and V. Hlupic, *Eds. Society for Modeling and Simulation International (SCS), October 2003,* pp. 97 – 103, delft, The Netherlands.

Mens, T., Czarnecki, K., and Van Gorp, P., A Taxonomy of Model Transformations, Electronic Notes in Theoretical Computer Science (ENTCS) Volume 152, March, 2006, pp. 125-142.

Mittal, S., Martín. J. L. R., Zeigler, B.P., « DEVSML: automating DEVS execution over SOA towards transparent simulators », Proceedings of the 2007 ACM Spring Simulation Multiconference, March 25-29, 2007, Norfolk, VA, USA, Vol. 2, pp. 287-295.

Mittal, S., Douglass, S.A., DEVSML 2.0: The Language and the Stack, DEVS Symposium, Spring Simulation Multiconference 2012, Orlando.

Mooney, J. and Sarjoughian, H.S., A Framework for executable UML models, In High Performance

Computing & Simulation Symposium, Spring Simulation Conference, pages 1-8, 2009.

Nikolaidou, M., Dalakas, V., Kapos, G.-D., Mitsi, L. and Anagnostopoulos, D., « A UML 2.0 profile for DEVS: *Providing code generation capabilities for simulation » in Proceedings of 16th International*.

Conference on Software Engineering and Data Engineering (SEDE-2007), Las Vegas, USA, July 2007 (Invited paper).

Nikolaidou, M., Dalakas, V., Mitsi, L., Kapos, G.-D., Anagnostopoulos, D. « A SysML Profile for Classical DEVS Simulators » (Conference Paper) Proceedings of the 2008 The Third International Conference on Software Engineering Advances, 978-0-7695-3372-8, Pp 445-450, 2008, 10.1109/ICSEA.2008.24, IEEE Computer Society.

OMG 2001. Model Driven Architecture homepage - http://www.omg.org/mda/.

OMG UML 2011. Unified Modeling Language specifications-http://www.omg.org/spec/UML/2.4.1

OMG SysML 2012. Systems Modeling Language - http://www.omg.org/spec/SysML/1.3/.

Posse, E., Bolduc, J.S., Vangheluwe, H., Generation of DEVS Modelling & Simulation Environments. In Proceedings of the *2003 Summer Computer Simulation Conference SCSC 2003*.

Risco-Martin, J.L., Mittal, S., Zeigler, B.P., Cruz, J.L, "From UML Statecharts to DEVS State Machines using XML", Multi-paradigm Modeling, *IEEE/ACM International Conference on Model Driven Engineering Languages and Systems , 2007.*

Rumbaugh, J., Jacobson, I. and Booch, G., "The unified modeling language reference manual", The Addison-Wesley object technology series, Addison-Wesley, 2005.

Sarjoughian H.S. and Markid, A.M., 2012. EMF-DEVS modeling. In *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium* (TMS/DEVS '12). Society for Computer Simulation International, San Diego, CA, USA.

Schulz, S., T. C. Ewing, and J. W. Rozenblit. 2000. Discrete event system specification (DEVS) and statemate statecharts equivalence for embedded systems modeling, Proceedings of the 7th IEEE *International Conference and Workshop on the Engineering of Computer Based Systems, Edinburgh,* Scotland, 2000.

Seo, C. "Interoperability between DEVS Simulators using Service Oriented Architecture and DEVS Namespace", Ph.D. dissertation, Electrical and Computer Engineering Dept., University of Arizona, Spring 2009.

SISO 2008. Simulation Interoperability Standards Organisation, SISO-REF-019-2008: Discrete-Event Systems Specification (DEVS).

http://www.sisostds.org/ProductsPublications/ReferenceDocuments.aspx.

Song, H., Infrastructure for DEVS Modelling and Experimentation. Master's thesis. McGill University. School of Computer Science. (2006)

Steinberg, F.D., Budinsky, F., Paternostro, M., and Merks, E. Eclipse Modeling Framework 2nd Edition, Addison Wesley, 2009.

Touraille, L., Traoré, M.K., Hill, D., "On the interoperability of DEVS components : On-Line vs. OffLine Strategies.", 2009, UMR CNRS 6158, LIMOS/RR-09-04, 13 p.

Touraille, L., Traoré, M.K., and Hill, D. R. C., 2011. A model-driven software environment for modeling, simulation and analysis of complex systems. In Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium (TMS-DEVS '11). Society for Computer Simulation International, San Diego, CA, USA, 229-237.

Zinoviev, D., "Mapping DEVS Models onto UML Models," Proc. of the 2005 DEVS Integrative M&S Symposium, San Diego, CA, April 2005, pp. 101-106.

Zeigler, B.P., Theory of Modeling and Simulation, New-York: Wiley-Interscience, 1976.