A Model-Driven Software Environment for Modeling, Simulation and Analysis of Complex Systems

Luc Touraille[12], Mamadou K. Traoré[12], David R.C. Hill[12]
[1] Clermont Université, Université Blaise Pascal, LIMOS, BP 10448, F-63000 CLERMONT-FERRAND
[2] CNRS, UMR 6158, LIMOS, F-63173 AUBIERE
touraille@isima.fr, traore@isima.fr, david.hill@univ-bpclermont.fr

Keywords: DEVS, Model Driven Engineering, interoperability, software environment

Abstract

SimStudio is a Modeling & Simulation environment based on the DEVS formalism (Discrete EVent Systems Specification). Its architecture aims at integrating in a single platform tools for modeling, simulation, analysis and collaboration, by proposing model transformation features (code generation, among others) in order to smooth the modeling and simulation cycle. To achieve this, SimStudio is built upon the DEVS formalism, recognized by many as a "universal" simulation formalism, and upon a unifying language for representing DEVS models. Models are at the heart of the infrastructure we propose, in line with the Model-Driven Engineering (MDE) approach, at the boundary between software engineering and simulation.

1. INTRODUCTION

Scientists have identified a set of issues present in the Modeling and Simulation (M&S) domain [CGTI 2005] [GSAT 2005]. These issues concern several aspects:

– The lack of interdisciplinary collaborations. Indeed, modeling teams and computer scientists often seem to work in parallel and not in collaboration, leading on the one hand to poorly crafted and rarely robust software, and on the other hand to applications that are disconnected from the actual needs of end users or not enough spread.

– The simulation of models more and more complicated, either because they approach more and more the complexity of the real world (multi-scales, finer granularity), or because they integrate more facets than before.

– The lack of large and perennial M&S systems. At the moment, there are only a few complete, stable and recognized software suites for realizing an M&S project.

The SimStudio project [Traoré 2008] aims at designing a software environment solving at least partially these issues. This platform let users develop models in a collaborative way, either through web or desktop applications, by giving them access to model repositories fed by the community. Thanks to model transformations and notably code generation, the creation of operational solutions is facilitated. SimStudio exploit the potential of both Model Driven Engineering and the DEVS (Discrete EVent System Specification) formalism, presented in section 2. Several works have already been conducted in this perspective, and are evoked in section 3. In addition to modeling and simulation components, SimStudio is also designed to include analysis and visualization features. Its architecture, outlined in section 4, is based on a plugin system which will allow the addition of modules and the integration of existing tools, via model transformations. The communications between these heterogeneous modules rely on a model representation language entitled DML (DEVS Markup Language) [Touraille et al. 2009], sketched in section 5.

2. THE DEVS FORMALISM

Based on system theory, the DEVS formalism [Zeigler et al. 2000] provides a sound basis for modeling and simulation of discrete event systems, but also for integrating heterogeneous models. Many variants exist, but only two are fundamental: Classic DEVS (C-DEVS) and Parallel DEVS (P-DEVS). These two formalisms describe the same class of systems; we focus our works on P DEVS because it makes the handling of simultaneous events more intuitive, and is more suited to parallel computations than C-DEVS. DEVS has been used for several years in many applications, and was applied to a broad range of domains [Kim et al. 2000] [Muzy et al. 2005] [Farooq et al. 2007] [Innocenti et al. 2009].

Representing a system in DEVS can be done in two ways [Zeigler et al. 2000]. The first one consists in designing an *atomic model*, which describes the system as an entity holding a state (S), exhibiting an autonomous behavior specified with internal state changes (δ_{int}) and a time advance function (ta), reacting to environmental stimuli, i.e. external events (δ_{ext}), and generating events as its state changes through time (λ). Exchanges with the environment are performed through input (X) and output (Y) ports, which receive or generate events with which values are associated. In addition to this, P-DEVS atomic models also specify the behavior to adopt when a conflict

arises between an internal event (autonomous state change) and an external event (response to the environment) (δ_{con}).

The second way to specify a system with DEVS is by defining a *coupled model*. Coupled models have input and output ports, and aggregate components that can be either atomic models or coupled models themselves. These components are connected by their output and input ports (couplings); thereby, events generated by a model become stimuli for others.

DEVS strength resides in its expressive power. Indeed, it was constructed in an incremental way starting from minimal specifications, by adopting at each step a higher abstraction level bringing additional knowledge, while at the same time demonstrating the morphisms with lower levels. It is therefore possible to prove [Zeigler et al. 2000] that there exists a DEVS model for every discrete event system. But we can go further: indeed, the DEVS 'protocol' can be used as a "universal" simulator or a simulation "common knowledge bus", enabling the coupling of models written in heterogeneous formalisms and according to different paradigms. Two methods exist to integrate a non-DEVS model in the DEVS framework: co-simulation and model transformation [Schmidt 2006]. The latter consists in converting non-DEVS models to the DEVS formalism in order to obtain model specifications in a uniform language. Vangheluwe represents the different possible transformations using a diagram of formalism transformations [Vangheluwe 2000].

3. RELATED WORKS

DEVS simulators being precisely defined by the formalism, their implementation is quite simple. Consequently, we can find many DEVS simulation libraries. Among them, DEVSJava [DEVSJava 2004] proposes several tools, for example a graphical interface for visual modeling. However, DEVSJava does not take into account the entire M&S cycle, and is specific to both DEVS and Java. Along the same lines, JDevs [Filippi et al. 2002] includes a graphical module for editing models, as well as a DEVS simulator and two visualization tools (2D and 3D).

Regarding the integration of heterogeneous models, we can distinguish between two approaches:

– An approach centered on models, which consists in standardizing model representation to make it possible to exchange and compose them easily, as we did with DML. Several works had already been performed in this direction, but were either too restrictive to exploit all of DEVS expressive power [Risco Martín et al. 2007] [Janoušek et al. 2006], or specific to a given programming language [Mittal et al. 2007a].

– An approach centered on simulators (co-simulation), where the standardization effort is focused on inter-simulators communications instead of models. This domain was explored by the USA Department of Defense which proposed HLA (High Level Architecture) [HLA 2000] as a solution to the simulator interoperability issue. This solution seems to be much used for military simulations that involve heterogeneous and geographically distributed simulators, but requires an important implementation effort to be put into action. A more flexible approach, using a service oriented architecture and more precisely web services, can be found in [Mittal et al. 2007b].

The Virtual Laboratory Environment project (VLE) [Quesnel et al. 2009] falls within the realm of multi-modeling, by allowing the user to specify its models in several DEVS variants (Cell-DEVS, DS-DEVS...) and to couple them using the co-simulation principle. The software environment includes graphical design and visualization modules. However, the collaborative aspects and the access to high performance computing resources are not taken into account.

James II [Himmelspach 2007] is a generic M&S framework where one can plug formalisms and simulators as Java classes. However, to our knowledge, this platform does not make it possible to couple models defined in heterogeneous formalisms. Moreover, the plugins are highly coupled to the architecture, limiting the seamless integration of existing tools.

Another quite complete M&S software environment is described in [Bonaventura et al. 2010]. CD++Builder is an Integrated Development Environment with goals very similar to SimStudio: graphical interface for specifying models, code generation, result analysis tools and so on. The specification of atomic models can be done either in C++ or using the DEVS-Graph graphical syntax. The authors defined several formats for storing models, events, simulation results and so on. The software environment is developed using some of the Eclipse Modeling Project tools, but without exploiting all the MDE potential of this framework.

Finally, in [Posse et al. 2003], the authors expose a model-driven solution with a mindset similar to the one we adopted. They define a DEVS metamodel using the AToM^3 tool [De Lara et al. 2002], a metamodeling and model transformation platform. From this metamodel, they generate a graphical DEVS modeling environment, and also specify the necessary transformations to generate Python code compatible with the PythonDEVS simulator.

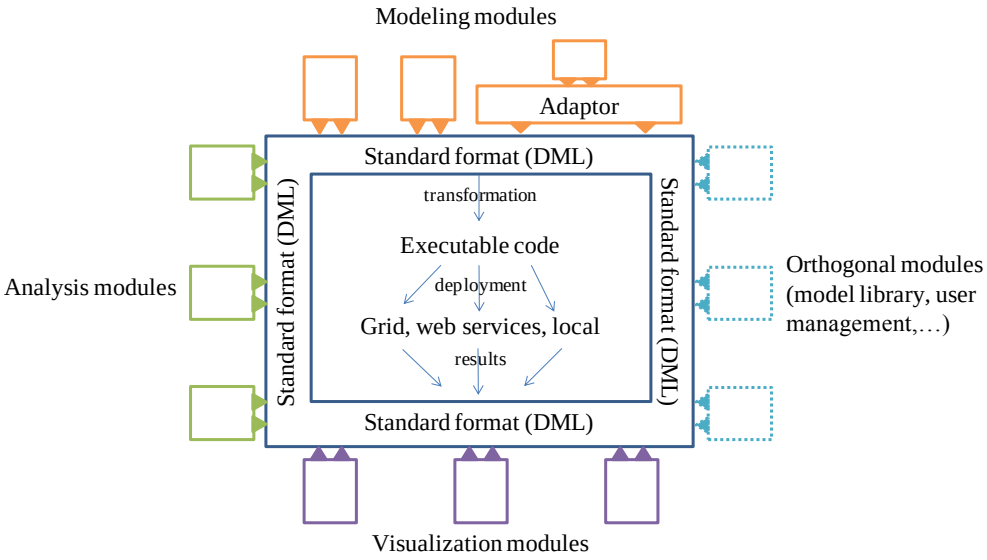

Figure 1. Overview of SimStudio architecture.

4. SIMSTUDIO ARCHITECTURE

Based on the DEVS formalism, the SimStudio environment aims at providing a complete M&S toolchain, which would assist the model developer from model design to results analysis. In order to incorporate the existing tools and make it possible to easily add new features, SimStudio uses a modular software architecture relying on plugins. The platform is constituted of a simulation kernel, on which are plugged software components that can be classified along several axes (Figure 1):

– A Modeling axis, which encompasses the graphical and textual tools for model design. "Adaptor" modules can also be provided to integrate existing modeling tool by transforming their output specification in a format understandable by SimStudio.

– An Analysis axis, with tools for formal analysis, statistical analysis, etc.

– A Visualization axis, handling the display of simulation results as charts, animations and so on, and possibly providing virtual reality features by letting the user interact with the simulation run.

– A Management axis, providing orthogonal services such as user management, model storage and querying, plugins configuration, etc.

The role of the simulation kernel is to automatically generate an operational solution from a model specification, then to handle its deployment and execution on various types of platforms, according to the user's choice. This can range from a simple execution offline or on a server to a distribution on a computing grid or cluster.

However, in order for these modules to communicate together, SimStudio must be based on a common language; we think that the DEVS formalism is the most adapted tool to fulfill this role, thanks to its expressive power. Indeed, the use of DEVS as a common denominator provides two important advantages:

1. The possibility to couple heterogeneous models, which can use different paradigms (continuous/discrete), different formalisms (differential equations, Petri nets, cellular automata...), different abstraction levels, etc.

2. The access to a rigorously defined operational semantic. DEVS strictly discriminate between models and simulators. As a consequence, a DEVS simulator is "universal" in the sense that it can simulate any DEVS model. By providing the model developers with the transformations needed to convert a model in a DEVS model, and the access to an efficient and configurable simulator, we avoid many repeated implementation efforts: the user can focus on modeling, without worrying about simulation.

Consequently, an essential prerequisite is the definition of a DEVS metamodel that is generic enough to support existing models and flexible enough to be usable in practice. Such a metamodel will enable a Model Driven Engineering (MDE) approach, in which the definition of new plugins (except for management modules) will boil down to the development of transformations from (resp. to) a DEVS model, conforming to the metamodel, to (resp. from) other models representing a range of views on the same system under study.

In the following section, we present the DEVS metamodel we developed, called DML (DEVS Markup Language) as proposed in [Hill 2000].

5. DML, A UNIFYING DEVS METAMODEL

The architecture of SimStudio is based on the use of a standardized representation of every aspect of DEVS models, defined by a metamodel. The latter must take into account a broad range of use cases, and must consequently propose a solution for specifying the following elements:

– DEVS model structure, i.e. state variables, input and output ports, and couplings.

– Atomic model dynamics, described by the various transition functions (internal, external and confluent).

– Model parameterization and initialization.

– Model behavior, corresponding to simulation results, represented by trajectories.

– Possibly graphical attributes, for model and results visualization.

In addition to this, in order to fulfill its unifying function, the metamodel must define a family of models independent of all platforms [Bézivin et al. 2008], a principle that can be found in the Model Driven Architecture (MDA is the Object Management Group MDE proposal) as the notion of Platform-Independent Model (PIM). Indeed, model developers must dispose of a "common language" to exchange and make their models interact, though they might be written for different simulators and in different programming languages. The principal difficulty consequently resides in representing the model dynamics: how can one describe an algorithm independently of all programming language, without at the same time depriving the model developer from all the libraries and functionalities peculiar to those languages?

Before answering this question, we must choose the metametamodel to which DML will conform. For standardization and interoperability reasons, we perform the communications between SimStudio components using XML (eXtensible Markup Language) [Abiteboul et al. 1999] documents. With the aim of rapidly developing a prototype, we specified our DEVS metamodel with XML Schema, because of its simplicity and the profusion of existing tools, e.g. for describing transformations with the Extensible Stylesheet Language Transformation (XSLT). As we will explain in section 6, we have since then adopted a framework more adapted to MDE, which provides more powerful model manipulation features.

In the next subsections, we present more precisely DML by addressing the requirements evoked previously.

5.1. Models Structure

DEVS models structure relies on the notion of set; this notion is easily captured in XML by the concept of type, largely used in programming. Input and output ports, state variables and parameters are consequently represented as type instances, which can be:

– intrinsic, i.e. supported by default, such as integers, real numbers or character strings;

– user-defined, using XML schemas. The model developer can create its own types to exploit them in his models. The XML schemas can also be generated from classes or structures, should the model be retro-engineered from existing code.

– library-dependent. In practical applications, the model developer may need to use certain types defined in libraries of a given programming language. To find a compromise between platform independence and the liberty needed by the model developer, we propose to use the notion of *abstract data types* (ADT) [Liskov et al. 1974]. Before being usable, these types must be bound to a specific implementation in the targeted programming language. For example, an ADT *RandomNumberGenerator* could be bound to the Java class `umontreal.iro.lecuyer.rng.MT199937` [L'Ecuyer et al. 2002] and to the C++ class `boost::mt19937`.

Code 1 shows a sample description for model state variables. This model holds two variables: *sigma*, a real number, and *buffer*, an instance of the *Queue* ADT. In order to generate the corresponding code in target languages, bindings between ADTs and concrete types must be provided. For this purpose, we define library models that indicate what types must be used to fulfill the role of given ADTs. Code 2 and Code 3 give the mapping of the *Queue* ADT in the Java Class Library and in the Standard C++ Library.

```
<state>
  <variable>
    <name>buffer</name>
    <type kind="libDependent">Queue</type>
  </variable>
  <variable>
    <name>sigma</name>
    <type>double</type>
  </variable>
</state>
```

Code 1. Description of state variables.

```
<type id="Queue">
  java.util.LinkedList<String>
</type>
```

Code 2. Extract from the "Java Class Library" model.

```
<type id="Queue">
  std::queue<std::string>
</type>
```

Code 3. Extract from the "Standard C++ Library" model.

These library models will be used during the transformations from DEVS models to operational solutions, enabling the automatic generation of a model specific to the target language (cf. the notion of Platform Specific Model (PSM) in MDA). Thus, instances of the *Queue* type will be linked lists of strings in Java, and queues of strings in C++. The various transformations necessary are detailed in subsection 5.4.

5.2. Representation of Algorithms

The approach we took to represent model dynamics is based on similar principles. Indeed, we need to represent the logic of model functions in a way that is independent from the platform, but we need to let the user free to use peculiar features of a language or a library. Consequently, we use to describe these functions a combination of generic code, which can be mapped directly in existing imperative languages, and of abstract code snippets that need to be implemented for each target language (cf. Code 4).

```
<if>
  <test>
    <binaryExpr op="equality">
      <lhs><var>sigma</var></lhs>
      <rhs><literal>0</literal></rhs>
    </binaryExpr>
  </test>
  <then>
    <codeSnippet kind="libDependent"
                 id=  "enqueue">
      <param id="queue">buffer</param>
      <param id="element">
        <codeSnippet kind="simDependent"
                     id=  "getValueOnPort">
          <param id="port">in</param>
        </codeSnippet>
      </param>
    </codeSnippet>
  </then>
</if>
```

Code 4. XML representation of a “semi-generic” code.

In this example, an element is added to the *buffer* if the *sigma* variable equals zero. The insertion into the queue cannot be described in a generic way since it depends on the concrete type used. For instance, in Java, the insertion will be performed with the *offer* method of *java.util.LinkedList*, whereas in C++, the method *std::queue::push* will be called.

Moreover, some parts of the code do not depend only on the target language but also on the target simulator. For example, fetching a value on an input port is done in many different ways in the various existing implementations. It is therefore necessary to permit the specification of simulator-dependent code snippets, like *getValueOnPort* in our example. Eventually, we use three independence levels, in relation to (i) programming languages, (ii) libraries and (iii) simulators. As previously, to obtain an executable specification, these independences need to be “resolved” using platform models and transformations.

Thanks to this solution, DML gives a great flexibility for writing models. Most existing DEVS models, which were developed in various DEVS tools, can be represented, and the quantity of code that needs to be written to port a model from a platform to another is limited to the minimum. Only ADTs and abstract code snippets need to be concretized in a specific implementation. This stage can even be simplified by collectively constructing “catalogs” of types and snippets along with their implementation in the most popular languages – these catalogs would in fact be a kind of platform models. Similarly, DEVS framework authors can publish their simulator model to facilitate the integration of their tool in SimStudio.

Besides being used to describe DEVS models transition functions, this “semi-generic” code is also used in DML to describe parameterization and initialization. Indeed, these two stages sometimes boil down to simple variables initializations, but can also be more complex (random number generation, file reading, etc.).

5.3. Simulation Results

The behavior of a DEVS model is entirely represented by all its potential input/output trajectories, meaning all the pairs (input trajectory, output trajectory) that the model can generate, an input (resp. output) trajectory being the set of events generated on input (resp. output) ports during a simulation run. However, it is often convenient to also observe the evolution of the model state variables, as well as the behavior of the sub-models in the case of a coupled model. All this simulation results are represented in DML as sets of values annotated by a timestamp denoting simulation time. These trajectories can then be exploited to analyze the behavior of the model, graphically render the simulation, or feed another model with data.

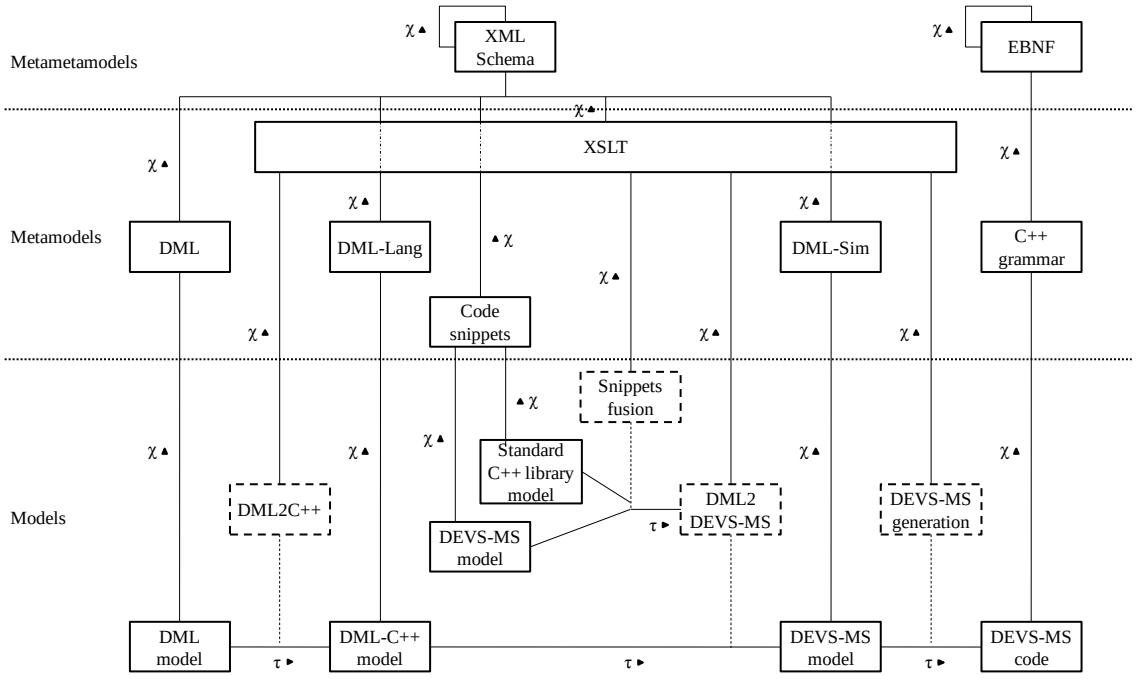

Figure 2. Models, metamodels, metametamodels and transformations involved in the SimStudio kernel.

5.4. Overview of the Conformity Relations and Transformations

In the previous paragraphs, we evoked several times the transformations needed to obtain an operational solution from a DML model. Figure 2 presents these various transformations, as well as the conformity relations between models, metamodels and metametamodels used in SimStudio. In this example, we transform some DEVS model into code understandable by the DEVS-MetaSimulator (DEVS-MS) [Touraille et al. 2010], the simulator we developed for the SimStudio kernel, written in C++.

We employ in this figure the notations used in [Favre et al. 2006]:

– χ means "conforms to";

– τ means "is transformed into".

On the other hand, we denote transformation models by dashed rectangles. They are connected by a dotted line to the relations "is transformed into" (τ) where they are used.
The best way to analyze this figure is to read it from bottom to top and from left to right. At the heart of the architecture reside DML models; they conform to the DML metamodel presented previously, itself conforming to the XML Schema metametamodel. The achievement of an operational solution is performed through successive refinements (transformations towards a target platform), until obtaining a source code for a given simulator.

As a first stage, we perform a transformation from a DML model to a model said "DML-Lang", specific to a target programming language. This transformation, described by an XSLT model, consists in replacing the "generic" code parts by their equivalent in the destination language. Elements such as assignments, conditional branches, loops, etc., are translated, as well as the specifications of user-defined types.

The "DML-Lang" metamodel is very similar to DML, except for the code elements that now contain mixed content (text and XML tags): the generic structure of functions has been generated, but the library and simulator-dependent snippets has not been transformed yet. This will be covered by a second transformation.

The latter involves two types of platform models: library models and simulator models, all conforming to the "Code snippets" metamodel and specifying mapping between abstract operations used in the DML model and concrete operations to be performed. Authors of DEVS tools must have the possibility to override mappings expressed in the library models, for example to use certain data structures integrated in their inheritance hierarchy. For this reason, we handle the two types of dependent snippets in a single step, thanks to a higher order transformation ("Snippets fusion") which merges the two transformation models into a new one ("DML2DEVS-MS"). Concretely, this transformation generates an XSLT model that applies the replacements defined by the two models (library and simulator) while

giving the priority to the ones specified for the simulator. The generated transformation can then be applied to the DML-Lang model to obtain a DML-Sim model. Issuing from this transformation, all the elements of the DML model representing code (notably all the transition functions) have been translated into the target language.

All is left to do is generated the complete code permitting the integration of the model into the target tool. Usually, this involves generating a class, defining input and output ports, couplings, etc. This must be done in respect with the constraints peculiar to the target framework: inheriting from a base class, defining certain attributes and so on. Since the operations needed can be quite disparate, we leave the DEVS simulators authors free to define their own transformation models conforming to XSLT.

Eventually, several model repositories will be available, providing:

– DML models, meant to be reused either in a standalone manner or as components in coupled models;

– library models, offering a large choice to the user;

– simulator models and associated transformation models, so as to enlarge the scope of DML to the maximum of existing tools.

6. CONCLUSION AND FUTURE WORKS

In this article, we presented the architecture of SimStudio, an M&S environment that encompasses in a single platform tools for modeling, analysis, simulation and visualization. Its aim is to accelerate the M&S cycle by (i) facilitating the conception of models thanks to modeling modules and model repositories; (ii) automating the generation of executable artefacts, in a variety of simulators, thanks to model transformations; (iii) integrating analysis tools into the platform to smooth the iterative process of conception/simulation/analysis.

Endued with a modular architecture, SimStudio enables the addition of new features and the integration of existing tools thanks to a Model Driven Engineering approach and the use of a generic and flexible DEVS metamodel.

The ideas presented in this paper have known quite an evolution recently, and many developments have been performed towards achieving SimStudio goals. They are still in progress, but are worth mentioning here. First of all, we now use a set of tools nicely suited for MDE, which are developed by the Eclipse Modeling Project (EMP) [Gronback 2009]. Indeed, EMP provides a range of tools for metamodeling, i.e. describing abstract syntax (Eclipse Modeling Framework), creating concrete syntaxes, either graphical (Graphical Modeling Framework) or textual (Textual Modeling Framework) and developing model transformations and text generators (Model to Model Transformation, Model to Text Transformation).

At the time of this writing, one of our master students is in the process of finalizing a graphical editor for DEVS models, using GMF and the DEVS-Driven Modeling Language [Traoré 2009]. We also started the specification in Ecore of the DEVS metamodel presented here, but it still lacks the ability to describe the dynamics of atomic models. Regarding the structure of DEVS models, the metamodel is operational and can already be used to specify models and generate DEVS-MS and CD++ code thanks to model-to-text transformations written with Xpand. We also experimented model-to-model transformations, using the Atlas Transformation Language, by generating an XHTML documentation from a model specification.

In the close future, we will carry on the implementation of the DEVS metamodel by including support for "semi-generic" code, as exposed in this paper. Another development will be writing the transformation from models produced by the graphical editor to models conforming to the DEVS metamodel. Finally, we will try to increase the number of supported existing tools, by including for example DEVSJava. In the long term, we plan to experiment with formalism transformations by writing model transformations from existing metamodels (e.g. Petri Net [Wachsmuth 2007]) to the DEVS metamodel.

References

Abiteboul S.; Buneman P.; Suciu D. 1999. *Data on the Web: From Relational to Semistructured Data and XML*. Morgan Kaufmann.

Bézivin J.; Vallecillo-Moreno A.; García-Molina J.; Rossi G. 2008. "MDA at the Age of Seven: Past, Present and Future." European Journal for the Informatics Professional, vol. IX, no. 2, April, pp. 4-7.

Bonaventura M.; Wainer G.A.; Castro R. 2010. "Advanced IDE for Modeling and Simulation of Discrete Event Systems." In Proceedings of the 2010 Spring Simulation Conference (SpringSim'10), DEVS Symposium, April 11-15, Orlando, Florida, USA, Article No. 125

CGTI – Groupe Conseil Général des Technologies de l'Information. 2005. "La politique française dans le domaine du calcul scientifique". Report no. II.B.14, March.

De Lara J.; Vangheluwe H. 2002. "AToM3: A Tool for Multi-formalism and Meta-modelling." In Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering, April 6-14, Grenoble, France, Lecture Notes In Computer Science, vol. 2306, pp. 174-188.

DEVSJava. 2004. ACIMS software site: http://www.acims.arizona.edu/SOFTWARE/software.shtml Last accessed May 2009.

Farooq U.; Wainer G.; Balya B. 2007. "DEVS Modeling of Mobile Wireless Ad Hoc Networks." Simulation Modelling Practice and Theory, vol. 15, no. 3, pp. 285-314.

Favre J.; Estublier J.; Blay-Fornarino M. 2006. *L'ingénierie dirigée par les modèles. Au delà du MDA*, Hermès – Lavoisier.

Filippi J.B.; Delhom M.; Bernardi F. 2002. "The JDEVS Hybrid Modelling and Simulation Environment." In Proceedings of the iEMSs First Biennial Meeting: International Congress on Environmental Modelling and Software (iEMSs 2002), International Environmental Modelling and Software Society, Lugano, Switzerland, vol. 3, pp. 283-288.

Gronback R.C. 2009. *Eclipse Modeling Project – A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley Professional.

GSAT - Groupe Simulation – Académie des Technologies. 2005. "Enquête sur les frontières de la simulation numérique en France. La situation en France et dans le monde. Diagnostic et propositions". Report from the French Technologies Academy, May.

Hill D. 2000. "Contribution à la modélisation de systèmes complexes : application à la simulation d'écosystèmes". French Research Habilitation Thesis, Blaise Pascal University.

Himmelspach J. 2007. "Konzeption, Realisierung und Verwendung eines allgemeinen Modellierungs-, Simulations- und Experimentiersystems - Entwicklung und Evaluation effizienter Simulationsalgorithme". PhD Thesis, Universität Rostock.

HLA; IEEE 1516-2000, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules." Institute of Electrical and Electronics Engineers, May 1.

Innocenti E.; Silvani X.; Muzy A.; Hill D. 2009. "A software framework for fine grain parallelization of cellular models with OpenMP: Application to fire spread." Environmental Modelling & Software, vol. 24, pp. 819-831.

Janoušek V.; Polášek P.; Slavíček P. 2006. "Towards DEVS Meta Language." In Proceedings of ISC 2006, June 7, Palermo, Italy, pp. 69-73.

Kim D.; Cao H.; Buckley S.J. 2000. "Modeling and Simulation of Supply Chain Management Based on DEVS and CORBA Framework." In Proceedings of the 2000 AI, Simulation and Planning in High Autonomy Systems Conference, March 6-8, Tucson, Arizona, USA, pp. 282-289.

L'Ecuyer P.; Meliani L.; Vaucher J. 2002. "SSJ: A Framework for Stochastic Simulation in Java." In Proceedings of the 2002 Winter Simulation Conference, December 8-11, San Diego, California, USA, vol. 1, pp. 234-242.

Liskov B.; Zilles S. 1974. "Programming with abstract data types." ACM SIGPLAN Notices, vol. 9, no. 4, pp. 50-59.

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

Mittal S.; Risco-Martín J.-L.; Zeigler B.P. 2007. "DEVS-based simulation web services for net-centric T&E." In Proceedings of the 2007 Summer Computer Simulation Conference, July 15-18, San Diego, CA, USA, pp. 357-366.

Muzy A.; Innocenti E.; Aïello A.; Santucci J.F.; Santoni P.A.; Hill D. 2005. "Modelling and simulation of ecological propagation processes: application to fire spread." Environmental Modelling & Software, vol. 20, Issue 7, July, pp. 827-842.

Posse E.; Bolduc J.-S. 2003. "Generation of DEVS Modelling & Simulation Environments." In Proceedings of the 2003 SCS Summer Computer Simulation Conference, July, Montréal, Canada, pp. 295-300.

Quesnel G.; Duboz R.; Ramat E. 2009. "The Virtual Laboratory Environment - An Operational Framework for Multi-Modelling, simulation and analysis of complex dynamical systems." Simulation Modelling Practice and Theory, vol. 17, April, pp. 641-643.

Risco-Martín J.-L.; Mittal S.; López-Peña M. A.; De La Cruz J. M. 2007. "A W3C XML Schema for DEVS Scenarios." In Proceedings of the 2007 ACM Spring Simulation Multiconference, March 25-29, Norfolk, VA, USA, vol. 2, pp. 279-286.

Schmidt D.C. 2006. "Model-Driven Engineering - Guest Editor's Introduction." IEEE Computer, February, vol. 39, no. 2, pp. 25-31.

Touraille L.; Traoré M.K.; Hill D.R.C. 2009. "A Markup Language for the Storage, Retrieval, Sharing and Interoperability of DEVS Models." In Proceedings of the 2009 ACM/SCS Spring Simulation Multiconference, March 22-27, San Diego, CA, USA, Article no. 163.

Touraille L.; Traoré M.K.; Hill D.R.C. 2010. "Enhancing DEVS Simulation through Template Metaprogramming: DEVS-MetaSimulator", In Proceedings of the 2010 ACM/SCS Summer Simulation Multiconference, July 11-15, Ottawa, ON, Canada, pp. 394-402.

Traoré M.K. 2008. "SimStudio: a Next Generation Modeling and Simulation Framework." In Proceedings of the ACM/IEEE 1st International Conference on Simulation

Tools and Techniques for Communications, Networks and Systems & Workshops, March 3-7, Marseille, France, Article no. 67.

Traoré M.K. 2009. "A Graphical Notation for DEVS." In Proceedings of the 2009 ACM/SCS Spring Simulation Multiconference, March 22-27, San Diego, CA, USA, Article no. 162.

Vangheluwe H.L.M. 2000. "DEVS as a Common Denominator for Multi-formalism Hybrid Systems Modelling." IEEE International Symposium on Computer-Aided Control System Design, September 25-27, Anchorage, Alaska, USA, pp. 129-134.

Wachsmuth G. 2007. "Metamodel Adaptation and Model Co-Adaptation." ECOOP'07. vol. 4609 of LNCS., Springer pp. 600-624.

Zeigler B.P.; Praehofer H.; Kim T.G. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press.

Biographies

Luc Touraille is a PhD student at the LIMOS CNRS laboratory under Dr. Traore and Dr. Hill supervision. He is working on the DEVS formalism and more particularly on applying Model Driven Engineering techniques to this domain. He holds a M.Sc. and a French "diplôme d'Ingénieur" in Computer Science & Software Engineering.

Mamadou K. Traoré obtained his PhD in Computer Science in 1992. He is Associate Professor at the Blaise Pascal University (Clermont-Ferrand - France) since 1995. His current research focuses on formal specification, symbolic manipulation and automated code synthesis of simulation models (http://www.isima.fr/~traore).

David R.C. Hill is currently Vice President of Blaise Pascal University in charge of ICT. Since 1990, Professor Hill has authored or co-authored papers in various domains with a particular focus on ecological modeling and life science simulation. More information can be found here: http://www.isima.fr/~hill.