VLE – A Multimodeling and Simulation Environment

Gauthier Quesnel
Institut de Recherche pour le Développement
Sète, France
gauthier.quesnel@ifremer.fr

Raphaël Duboz
Institut de Recherche pour le Développement
Sète, France
duboz@ird.fr

Éric Ramat
Laboratoire d'Informatique du Littoral
LIL UPRES EA 4029
Calais, France
ramat@lil.univ-littoral.fr

Mamadou K. Traoré
Laboratoire d'Informatique de Modélisation et d'Optimisation des Systèmes
LIMOS CNRS UMR 6158
Aubière, France
traore@isima.fr

Keywords: Multimodeling, Modeling and Simulation Cycle, DEVS Extensions

Abstract

Modeling and Simulation (M&S) is becoming the core of scientific activities addressing nature complexity. To tackle complexity, we need to integrate heterogeneous formalisms in the same model. Models then become multimodels. Furthermore, M&S softwares have to assist take into account the M&S activity cycle, i.e. modellers to design and to implement models, to define experimental frames and to analyse simulation results. Therefore, the M&S field needs reliable softwares oriented towards multimodels design and execution. The aim of this article is to introduce the Virtual Laboratory Environment (VLE). VLE is a both a software and an API which supports multimodeling and simulation. It addresses the reliability issue by using recent developments of the theory of M&S proposed by Zeigler. We present VLE in the context of the M&S cycle.

1. INTRODUCTION

The increasing complexity in systems modeling and Simulation (M&S) need reliable softwares to be addressed. Indeed, research fields such as ecology or sociology for instance, use M&S softwares to better understand dynamics of systems they focus on. The main difficulty for these disciplines is to integrate heterogeneous knowledges in a unique representation. Heterogeneous knowledges are supported by heterogeneous languages or formalisms. Therefore, M&S tools addressed to complex systems modeling must be able to take into account this heterogeneity. Some works refer to the modeling of heterogeneous systems as multiparadigm modeling [19] or multimodeling [6]. The multimodeling methodology is well defined in [5]. Works undertaken since the years 1970 and initiated by Zeigler [23, 24] define a formal framework that we can use for multimodeling. We recall some basis of these works in the first part of this paper.

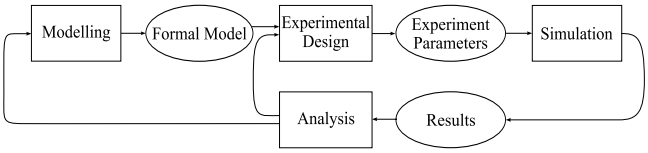

Figure 1. modeling and simulation cycle: Boxes are actions and ellipses are products of actions.

Another important aspect of M&S tools is to respect the classical M&S cycle as presented figure 1. Indeed, the modeller must be able to design his model. This model must be specified using one or several formalisms. Then the modeller has to construct an experimental frame. This experimental frame is introduced as a context for the simulator of the model, providing results to be analyzed. This is a cycle since we can reiterate these actions from analysis by modifying either the experimental frame or the model design.

We assume that the integration of heterogeneous models and the respect of M&S cycle are the fundamental key issues to provide a complete and reliable software for complex systems studies. Our works have started with multidisciplinary issues, mixing computer scientists and biologists. These works leaded to a first version of the Virtual Laboratory Environment (VLE) [15]. In this paper, we present recent advances in VLE. The first implementation of VLE did not consider multimodeling nor M&S cycle. We think that VLE is now mature enough to deal with a large number of complex systems.

We start by introducing the fundamental basis of our works. Then, we describe the concrete architecture of VLE in the context of the M&S cycle. We close this paper by a discussion and some opened works.

2. BACKGROUND

2.1. Formal Background

Our works take place in the M&S theory defined by Zeigler [23]. M&S theory tends to be as general as possible. It ad-

 ISBN # 1-56555-316-0

dresses major issues of computer sciences. From artificial intelligence to model design and distributed simulations, M&S theory aims to develop a common framework (formal and operational) for the specification of dynamical systems. In this section, we focus on DEVS formalisms and associated extensions. Many theoretical basis and formal extensions to DEVS were carried out, therefore, we advise the second edition of B.P. Zeigler's book [24] to have an overall picture of those works.

VLE is based on the Discrete Event System Specification (DEVS)[23]. In addition to classic DEVS models, VLE supports four DEVS extensions:

1. The Quantified State System (QSS1 and QSS2) embedded in DEVS [10] for the M&S of continuous systems.

2. The Dynamic Structure DEVS (DSDEVS) [1] for the M&S of systems where drastic changes of structures and behaviors occurred.

3. The Cellular automata DEVS (CellDEVS) [22] for the M&S of spatialized systems.

4. The CellQSS specification [20], a merge of CellDEVS and QSS extensions for the integration of spatialized differential equation systems.

Staying in a "DEVS world", we ensure the compatibility of models at formal and operational levels. Then, we can couple simulators with a well known algorithmic. We discuss this point in the following.

2.2. Operational Background

In this section, we deal with the operational integration of heterogeneous models. We do not give algorithms rather that describe the general framework. Algorithms can be found in the referenced literature. Here, we want to recall that it is possible to couple heterogeneous models using DEVS abstract simulators and then to build an operational framework for multimodeling.

We can address multimodeling or multiparadigm modeling following three orthogonal directions:

- The integration of several formalisms in a new one.
  We can cite Vangheluwe [18] who introduced the DEA (Differential Algebraic Equations), Zeigler [24] with the DEV&DESS and Barros [2] with the Heterogeneous Flow System Specification (HFSS) for the integration of continuous and discrete time systems.

- The specification of all sub-systems in one unique formalism.
  This approach implies to rewrite all sub-models using one common formalism.

- The co-simulation approach.
  Every sub-model has its own simulator specific to its own formalism. The main difficulty here is to couple these simulators.

Our works consider the second and the third aspect of multimodeling. Vangheluwe [18] has proposed DEVS as a common denominator for multimodeling. Indeed, it was shown that discrete time systems, Petri net and state charts, can be considered as DEVS sub-formalism [24]. Then, when possible, we use DEVS, QSS, DSDEVS or CellDEVS frameworks for models specification and simulation. It is not always possible to adopt a common formalism for all sub-models of the system. Therefore, we adopt the concept of the DEVS-Bus introduced by Kim [9]. DEVS-Bus enables interoperating with diverse discrete event modeling formalism. Any formalism to be integrated must be wrapped in a DEVS form through a suitably defined simulator. The VLE framework uses the wrapping concept [12]. In the following section, we present the concrete architecture of the VLE framework.

3. THE VIRTUAL LABORATORY ENVIRONMENT: VLE

As we have said before, VLE is oriented towards the integration of heterogeneous formalisms. Furthermore, VLE is able to integrate specific models developed in most popular programming languages into one single multimodel. To achieve that, VLE provides a complete library and graphical tools for models design and simulation. In the following, we introduce the VLE architecture and Application Programming Interface (API).

3.1. General framework

There are four interdependent applications in VLE, each one using a set of particular components (i.e. plug-ins):

- GVLE is a graphical user interface. It provides tools to visually construct a hierarchy of coupled models. Furthermore, a modeling plug-in can be use to define and to modify the behaviours of atomic models displaying a text editor where DEVS functions can be coded.
  GVLE enables the parameterization of experimental frames through the definition of experimental plans.

- EOV, *the Eyes Of VLE*, is a graphical application which observes the states of models at run-time of simulation. EOV is a set of visualization plug-ins. Every plug-in correspond to a type of display like colored grided surfaces or curves.

- VLE is the core of the environment. The three other applications depend on VLE (that is why the name of this application is the same as the general framework). VLE

implements the DEVS abstracts simulators and the extensions cited previously. To perform simulations, VLE main program load the experimental frame generated by GVLE (see next section) and then dynamically loads simulation components and visualisation components of EOV. Simulation plug-ins simulate the behaviours of the DEVS atomic models and VLE coordinates the simulation.

- AVLE (Analysis for VLE) is a graphical interface binding the experimental frame defined by GVLE and the R statistical tool [14].

All plug-ins are connected to the applications using a memory transfer or a TCP/IP protocol. Therefore, it is possible to develop components in any programming languages and operating systems. The format of data exchange between the four applications is an eXtensible Markup Language (XML[1]) application. It describes the structure of atomic and coupled models, the experimental frames and the parameterization of visualization tools.

The VLE framework provides the developers a complete and portable Application Programming Interface (API) (figure 2) written in C++ [16]. C++ is a standardized[2] and ensure compatibility with a large number of operating systems and interoperability with major programming languages (Java, Fortran or Python for instance).

We use the portable libraries provided by the GNU Project Gnome [7, 8] (the graphical toolkit GTKmm and the XML parser library libxml2 for instance). The choices of C++, the concepts of components and the Gnome libraries, enable to build an efficient and portable framework, easily modifiable and fast to develop.

3.2. Distributed simulation

VLE framework enables distributed simulation into a cluster of computers. It couples several simulators within a DEVS-Bus architecture and uses the algorithms of Parallel DEVS (PDEVS) for the coordination of models' simulators [24]. VLE does not natively support the parallelization of one isolated simulator. Nevertheless, this can be done specifically by a programmer for a particular atomic model. In VLE, distributed simulation is possible in two ways:

1. *Distributed models*: Every simulators and coordinators of a particular experiment are distributed over several computers. In this case, the communications between distant simulators use a network XML protocol (figure 3). This protocol is available for several programming languages and platforms by using the SWIG [17] software[3].
2. *Distributed experiment instances*: It is possible to distribute experiment instances. Indeed, we often have to perform experiments with different parameter values. In addition, it can be useful to repeat several instances of a single experiment because of the stochasticity of models. Therefore, VLE provides the description of simulation instances (replicas) through its XML application.

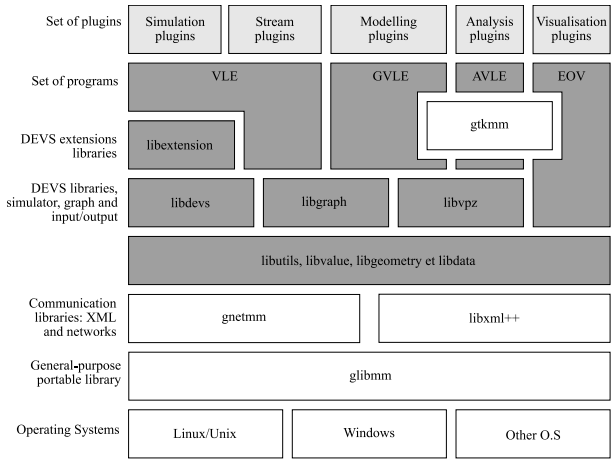

Figure 2. Representation of the VLE framework Application Programming Interface (API). Dark grey boxes are the libraries that we have developed. Clear grey boxes are model specific plugins developped by users and white boxes are external libraries coming from the GNU project to increase the portability.

The first type of distributed simulation we use permits to run simulators dedicated to a particular operating system and to couple it with other simulators. The main limitation here relies on message passing. Indeed, if models perform intense communications, it can be very time consuming for the simulation. The second type of distributed simulation depends on the number of available processors. It is not distributed simulation strictly speaking but it makes the execution of the experimental frame faster by distributing the replicas.

VLE is published under the free software licence GPL (GNU General Public License available at http://www.gnu.org/copyleft/gpl.html), thus allowing to copy, execute, modify and publish the software. The VLE environment is downloadable at http://vle.univ-littoral.fr.

[1]The main issue of XML is to facilitate the sharing of data across different systems. XML applications are described in a formal way, allowing some programs to modify and validate documents using these languages without prior knowledge of their content. A complete description of the XML language can be found on the W3C website or in several books [4].

[2]C++ has been standardized by ANSI (The American National Standards Institute), BSI (The British Standards Institute), DIN (The German national standards organization), ISO (The International Standards Organization) and several other national standards bodies.

[3]SWIG is a software development tool that reads C or C++ headers and generated wrapper code to make C/C++ accessible from other languages like Python, Java, C# etc. Nowadays, it supports around 19 programming languages.

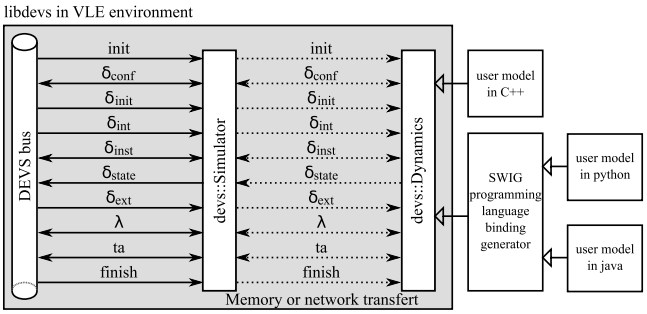

Figure 3. In this figure, we show how the user's model is connected to the DEVS-Bus of VLE (the gray box). The DEVS simulator is implemented with two classes (`devs::Simulator` and `devs::Dynamics`) to allow distributed simulation. We use the SWIG program [17] to build the bindings for different programming languages.

4. VLE AND THE MODELING AND SIMULATION CYCLE

In this section, we describe the M&S activity using VLE and referring to the M&S cycle described figure 1. Doing that, we go deeper within the VLE framework architecture showing how the modeller can use it.

4.1. Modeling

GVLE sustains the modeling activity. It is the Graphical User Interface (GUI) for VLE. GVLE modeling components are represented by pictures in the GUI. One component is associated with one atomic model. Using this interface, the modeller can couple models together. The resulting model is called a coupled model represented as an atomic model (a black box[4]). Doing that, the modeller can construct a hierarchy of coupled models. There is one window per coupled model. The whole hierarchy is shown in the right window as illustrated by the figure 4. There is at least one coupled model, the "top model".

The modeling components or plug-ins share informations about the dynamic of models they represent. They define which simulation components must be used and some other features specific to particular simulation components. For instance, the ordinary differential equation wrapper [12] component shown figure 4 specify the number and the form of equations to be used.

In addition to predefined modeling components, the modeller can use a more generic component that represents a generic atomic DEVS model. He can use this component to indicate a particular dynamic he has embedded in DEVS

As we have said in section 3.1., we use an XML application for data exchange between components and core programs. Results of the modeling activity described are stored in a VPZ (Virtual laboratory Project Zip) file. This latter can be generated by GVLE or by editing XML application in any text editor.

[4]Remain that the DEVS property of closure under coupling ensures that a coupled model is equivalent to an atomic one [24].

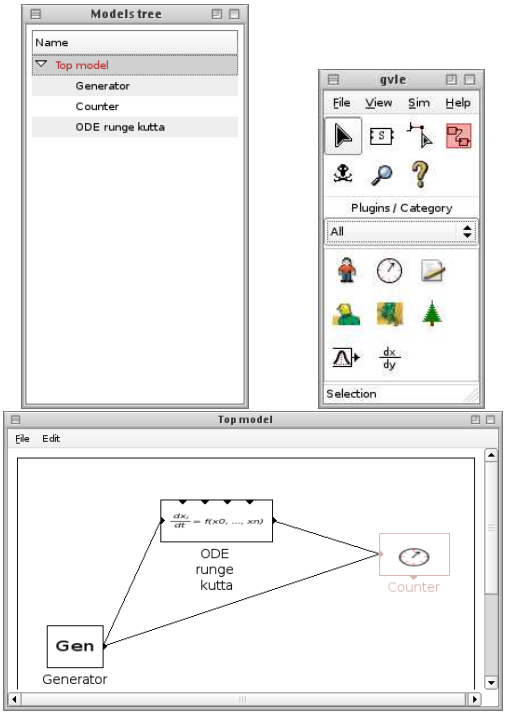

Figure 4. GVLE is used for the modeling and experimental design. It provides graphical user interfaces to manipulate atomic and coupled models. Icons on the top right represent tools for models management. The window at the center is a view of coupled model and the top left window shows the hierarchy of coupled models.

It can be very time consuming building complex coupled models. Furthermore, if there is a large number of models and complex connections between them, it can be difficult to simply modify the whole structure. To help the modeller, we have introduced the concept of *Translator* in VLE. The objective of the *Translator* component is to translate a particular XML application in a VPZ formated file. Doing that, the modeller can define a simple syntax to dynamically modify the structure of her/his model.

For instance, if we need to build a CellDEVS model, we must describe models, ports, connections and dynamics for each cells. The modeller can define an XML application as follow:

```
<cell rows="20" columns="20" dynamics="lifegame">
  <init>
    <cell x="2" y="5" value="false" />
    <cell x="3" y="5" value="false" />
    <cell x="4" y="5" value="false" />
  </init>
```

```
</cell>
```

In this example, the modeller creates a 20×20 cellular automata of the life game with particular initial conditions. The *Translator* parses this file and generates the corresponding VPZ file. It is then very easy to change the dimension or initial condition of the system.

Another possibility of using a *Translator* is the translation of any other format of model structure in the VPZ format. This open VLE to the use of any modeling tools. Nevertheless, building a translator component is a complex task and require knowledges in the DEVS formalism and the VLE XML application. As we have seen in the simple example above, the *Translator* must take into account both the model structure and the experimental frame (initial conditions etc.). This increases the complexity of its definition. In the following section, we discuss the construction of experimental frame using GVLE.

4.2. Experimental design

The VLE XML application (see section 3.1.) enables the definition of experimental frames. This construction is named "experimental design" i.e. the sates to observe and how to look at their evolution over time.

The modeller defines the initial conditions, the state variables to observe, the visualization components to use, the duration of the simulation and the execution mode, local or distant. In addition, the modeller can build an experimental plan. This latter is the definition of the variation domain for parameters and initial conditions and then the deduction of the number of simulations needed to achieve the experimental plan.

In VLE, we have define two particular types of model ports, initialization and state ports. The first are used for sending initial conditions, the second for sending state value to visualization components. Figure 6 gives an illustration of the use of state ports in VLE. Outputs can be treated by visualization components or stored directly in VLE. The outputs are first sent to one or more measure objects. A measure object can be attached to several different models.

The measures can be triggered in two ways:

- The modeller defines a time step and the simulator inserts the measure events in the simulator scheduler.
- When an internal or external transition occurs, the associated simulator receives measure events at the same time. The measure event is raised just after the transition event.

We consider that measure events do not influence the model simulator. This can occur for timed measures. To assert there is no modification of models' states when events of measure occur, the function managing these events has the following prototype:

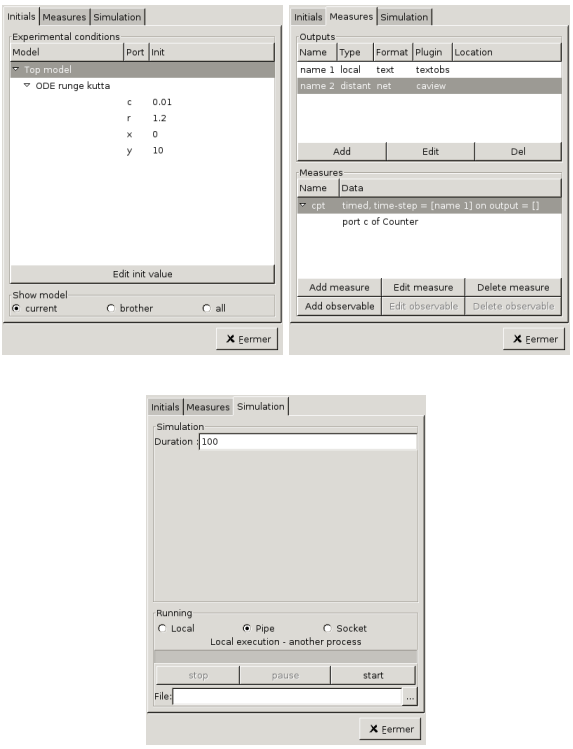

Figure 5. Definition of experimental frame. The top window on the left defines initial conditions. The top right window defines the state variables to observe and the visualization component to use. The window at the bottom defines the duration of the simulation and the execution mode.

```
Value processStateEvent(const StateEvent&) const;
```

The C++ "const" keyword at the end indicates that this function can not modify the model's states.

The policy for building experimental plans and the number of replicas can be defined in the VLE manager. The default one is an exhaustive experimental plan (the cross products of initials conditions).

The structure of the model and the experimental design activity are stored in a VPZ file. The modeller can then reproduce exactly the same experimental frame if needed. Once the modeling and experimental design activities have ended, the VPZ file contains all necessary informations needed by the VLE core program to perform simulations. That is the purpose of the next section.

4.3. Simulation

The VLE core program performs simulations. It first parses the VPZ file. Doing that, it gets all the necessary informations in concern with models structures, dynamics and the experimental frame. Thereafter, the VLE core program loads the useful simulation and visualization components and builds

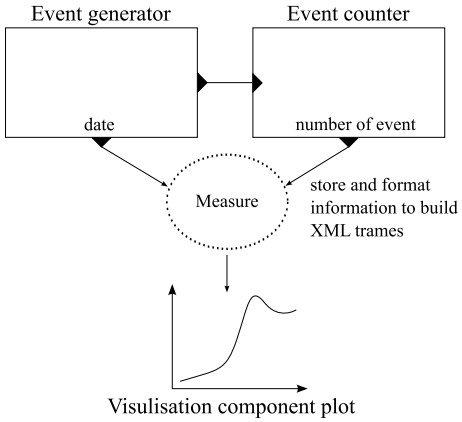

Figure 6. The diagram presents an example of the output management in VLE. It shows a coupling between an event generator (left model) and a counter (right model). Each model have one state port. The state ports are connected to a measure object that send values to the visualization component using network or memory transfer.

the coupled models network. Then, it sends an initialization message to every models and starts the simulation.

VLE implements DEVS abstract simulators. The DEVS extensions that we use in VLE implement their own abstract simulators. In order to schedule events between simulators, VLE uses the Parallel DEVS technique [24]. We have implemented a risk-free (i.e. conservative) algorithm for the Parallel DEVS simulation. This algorithm uses a global minimum synchronization time and performs collections of simultaneous output and events distribution.

All atomic models inherit the `Dynamics` class to build simulation component. The `Dynamics` class is a functional interface to bind coordinators and users' models. The following `Dynamics`'s functions can be overloaded by the user:

```
1 Time init();
2 void processInternalEvent(const InternalEvent&);
3 void processExternalEvent(const ExternalEventList&);
4 Time getTimeAdvance();
5 void getOutputFunction(const Time&, EventList& out);
6 void finish();
```

In addition, the Parallel DEVS function is implemented as follows:

```
1 type processConflict(InternalEvent&,
2                       ExternalEventList&) const;
```

This function is called when events occur at the same time. This function can be defined to choose the order of treatment for events between internal and external events.

The two functions dedicated to the realization of the experimental frame are:

```
1 void processInitEvent(const InitEventList&);
2 Value processStateEvent(const StateEvent&) const;
```

The first one processes the initialization events. The second one gives the current state of a model at a specified time. In the following section, we show how the measures performed on a model are returned to the modeller for analysis.

4.4. Analysis

Analysis is in concern with the interpretation of simulation results. VLE does not analyze data itself. As we have said in paragraph 3.1., the EOV application enables VLE to communicate with the R statistical software. Furthermore, VLE provides the analyst some visualization and storing components for data saving.

Visualisation components draw model states evolutions at run-time. Nowadays, the VLE framework proposes the following *Visualisation components*:

- The two-dimensional discrete component to draw discrete spatialized data (CellDEVS or CellQSS models for instance).
- The two or three-dimensional continuous space data component [5].
- The plot component, to draw curves and histograms (see figure 7).
- The gauge component to show values between a minimum and a maximum.

A *Visualization components* can use a *parametrization component*. The latter enables the analyst to customize the former (figure 7).

Visualization components can be developed by modellers using inheritances of specific GVLE classes. The *Visualization components* receive a parameterization message (in XML format) at the beginning of simulation.

The use of synchronous coordination between simulators and *Visualization components* can dramatically slow down the simulation. Therefore, VLE proposes the UNIX pipe technique to separate simulation from data management. Data are first encapsulated in an XML format, thereafter they are sent in a stream. Then, the visualization component receives the data and uses them. Data can also transit through the network. Therefore, the *Visualization components* can be located in another computer than the one used for simulation. A consequence of this is that *Visualization components* can be written in any language.

VLE proposes to directly store data in text files. Two formats are possible. The first one records one line per date and one column per saving state. The second one uses the Extended Markup Language, format for Statistical Data (StatDataML). This format is compatible with statistical applications like R/Splus, MATLAB, Octave. Then, the analyst can

[5] The 3D visualization components use OpenGL libraries

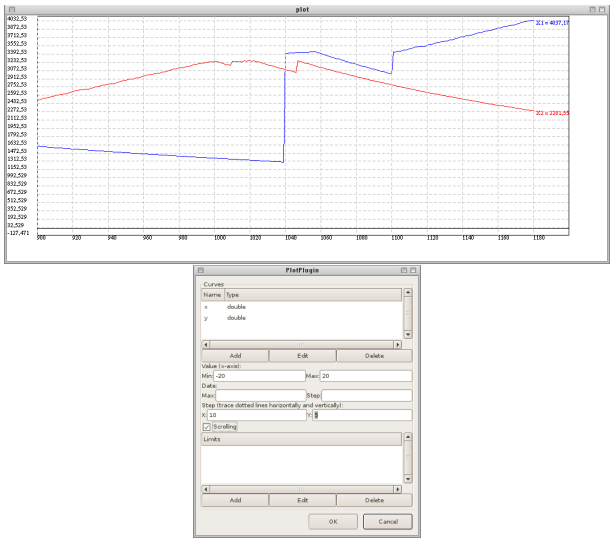

Figure 7. An example of *visualisation component* for curves. The right window shows the *parametrization component* useful for specifying the curves' labels and colors for instance.

directly load the simulation results in such statistical applications.

5. CONCLUSION AND PERSPECTIVES

In this paper, we have presented the VLE framework. It is a complete and powerful DEVS environment for the modeling and simulation of complex systems. We have mainly focused on the need for such a tool to be able to take into account the modeling and simulation cycle (figure 1). We have shown how the general architecture of VLE can support this cycle. Complex systems can better be addressed considering multimodeling as a basis for there construction. Therefore, VLE offers tools to couple heterogeneous models and to simulate them. As VLE is based on the M&S theory defined by Zeigler [24], it has a formal basis. This is a very important feature considering scientific uses of M&S. There are important classes of models that are not well suited to be specified as extension to DEVS. Nevertheless, many models can be wrapped inside DEVS models.

We have compared some platforms based on the DEVS formalism and VLE (ADEVS, CD++, DEVS/C++, DEVS/Java, JDEVS). This comparison is based on the list provided by G. Wainer. References to these platforms can be found at http://www.sce.carleton.ca/faculty/wainer/standard/tools.htm. We do not compare the whole of platform's characteristics rather than focus on the main features of VLE: i.e. the implementation of major DEVS extensions, experimental frames and the ability to perform distributed simulation. Considering this comparison and to our best knowledge, VLE appears to be a relatively complete DEVS environment.

VLE is an open tool. Indeed, it provides XML applications that permits to communicate with other frameworks. For instance, experimental frames and model structures are stored in an XML application. Then, any other simulation tool understanding this format is able to construct and simulate the specified model, under precise experimental conditions. This is a fundamental issue in complex scientific M&S activities to be able to reproduce and compare models. The concept of *Translator* we have briefly introduced in this paper seems very promising. We are developing an XML application to define a complete specification for MultiAgent Systems based on [3, 11].

Currently, the VLE Environment is used in two projects of majors French research institutes, Ifremer and IRD[6] and funding by the ANR Agency (National Research Agency). In the CHALOUPE project[7] VLE supports a model for the simulation of the dynamic of exploited marine biodiversity and the viability of fisheries. In the REMIGE project, VLE is used to model the impact of climate changes on marine top predators. Moreover, recent papers in concern with VLE have been published in international conferences [21, 13, 12]. Informations concerning VLE, including sources, examples, models and documentations are available on the VLE website[8].

REFERENCES

[1] F. J. Barros. Dynamic structure discret event system specification: Formalism, abstract simulators and applications. 13(1):35–46, 1996.

[2] F. J. Barros. Dynamic structure multiparadigm modeling and simulation. *ACM Transactions on Modeling and Computer Simulation*, 13(3):259–275, 2003.

[3] R. Duboz, E. Ramat, and G. Quesnel. Systèmes multiagents et théorie de la modélisation et de la simulation : une analogie opérationnelle. In Olivier Boissier et Zahia Guessoum., editor, *Actes des douzièmes Journées Francophones sur les Systèmes Multi-Agents (JFSMA) - Systèmes multi-agents défis scientifiques et nouveaux usages*, Paris, Novemver 2004.

[4] W. S. Means E. R. Harold. *XML in a Nutshell, 2nd Edition*. O'Reilly, 2002.

[5] P. A. Fishwick. *Simulation Model Desing and Execution*. Prentice Hall, 1995.

[6] P. A. Fishwick and B. P. Zeigler. A multi-model methodology for qualitative model engineering. *ACM transaction on Modeling and Simulation*, 2(1):52–81, 1992.

[6] http://www.ifremer.fr and http://www.ird.fr
[7] http://www.projet-chaloupe.fr
[8] http://vle.univ-littoral.fr

[7] Free Software Fundation. GNU Operating System: GNU's Not Unix, 1984. http://www.gnu.org.

[8] Free Sofware Fundation. Gnome: GNU Network Object Model Environment, the Free Software Desktop Project. http://www.gnome.org.

[9] J. Y. Kim and T. G. Kim. A heterogeneous simulation framework based on the DEVS bus and the High Level Architecture. In *Winter Simulation Conference*, Washington, DC, 1998.

[10] E. Kofman. A second order approximation for devs simulation of continuous systems. *Journal of the Society for Computer Simulation International*, 78(2), 2002.

[11] G. Quesnel. *Approche formelle et opérationnelle de la multi-modélisation et de la simulation des systèmes complexes – Apports pour les systèmes multi-agents*. PhD thesis, Université du Littoral Côte d'Opale, decembre 2006.

[12] G. Quesnel, R. Duboz, and É. Ramat. DEVS wrapping: A study case. In *Proceedings of CMS 2004 conference*, pages 374–382, Genoa, Italy, October 2004.

[13] G. Quesnel, R. Duboz, D. Versmisse, and É. Ramat. DEVS coupling of spatial and ordinary differential equations: VLE framework. In *Proceedings of OICMS 2005 conference*, Clermont Ferrand, France, June 2005.

[14] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2006. ISBN 3-900051-07-0.

[15] E. Ramat and P. Preux. Virtual laboratory environment (VLE): a software environment oriented agent and object for modeling and simulation of complex systems. In *Simulation Modelling Practice and Theory*, volume 11, pages 45–55, 2003.

[16] B. Stroustrup. *The C++ Programming Language*. Addison Wesley, 1986.

[17] Swig Development Team. Simplified Wrapper and Interface Generator, 1995. http://www.swig.org.

[18] H. Vangheluwe. Devs as a common denominator for hybrid systems modelling. In A. Varga, editor, *IEEE International Symposium on Computer-Aided Control System Design*, pages 129–134, Anchorage, Alaska, 2000. IEEE Computer Society Press.

[19] H. Vangheluwe, J. Lara, and P. J. Mosterman. An introduction to multi-paradigm modelling and simulation. In F.J. Barros and N. Giambiasi, editors, *AIS'2002. Simulation and Planning in High Autonomy Systems*, pages 9–20, Lisbon, Protugal, April 2002. Society for Modelling and Simulation International.

[20] D. Versmisse and E. Ramat. Management of perturbations within a spatialized differential equations system. In *European Simulation and Modelling Conference*, pages 520–524, Porto, Portugal, october 2005. SCS.

[21] D. Versmisse, J. C. Soulié, and G. Quesnel. Une étude de cas dans le domaine des pêcheries pour la simulation de systèmes complexes. In *Actes de la 6ème Conférence Francophone de Modélisation et Simulation Modélisation, Optimisation et Simulation des Systèmes : Défis et Opportunités*, Rabat, Maroc, april 2006.

[22] G. A. Wainer and N. Giambiasi. Application of the Cell-DEVS paradigm for cell spaces modelling and simulation. In *Simulation*, volume 76, pages 22–39, 2001.

[23] B. P. Zeigler. *Theory Of Modeling and Simulation*. Wiley Interscience, 1976.

[24] B. P. Zeigler, D. Kim, and H. Praehofer. *Theory of modeling and simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000.