Simulation Modelling Practice and Theory 17 (2009) 641–653

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

The Virtual Laboratory Environment – An operational framework for multi-modelling, simulation and analysis of complex dynamical systems

Gauthier Quesnel [a,*], Raphaël Duboz [b], Éric Ramat [c]

[a] INRA, UR875 Biométrie et Intelligence Artificielle, F-31326 Castanet-Tolosan, France
[b] CIRAD, UPR AGIRs 22, Campus International de Baillarguet, 34398 Montpellier Cedex 5, France
[c] ULCO, UPRES EA 4029 Laboratoire d'Informatique du Littoral, 50 Rue Ferdinand Buisson, 62228 Calais, France

A R T I C L E   I N F O

Article history:
Received 19 February 2008
Received in revised form 18 August 2008
Accepted 13 November 2008
Available online 10 December 2008

Keywords:
Multi-modelling and simulation
DEVS
Complex systems
Agent based model

A B S T R A C T

The cross-disciplinary activity of modelling and simulation is the core of the scientific activities addressing the complexity of nature. In this context, we need reliable computational environments to integrate heterogeneous representations coming from different scientific fields. Therefore, such environments must be able to integrate heterogeneous formalisms in the same model and assist the modeller for the design and implementation of models, the definition of the experimental frames and the analysis of simulation results. The aim of this article is to introduce a tool supporting all these features, the Virtual Laboratory Environment (VLE). VLE is a software and an API which supports multi-modelling, simulation and analysis. It addresses the reliability issue by using recent developments in the theory of modelling and simulation proposed by Zeigler. We present VLE in the context of the modelling and simulation cycle and show the effectiveness of the tool with a multi-model of fireman fighting a fire spread.

1. Introduction

A large part of the scientific activity is oriented toward the integration of cross-disciplinary knowledges. This is particularly true in the Environmental Sciences (ES), where disciplines such as physics, biology, ecology and economy are the different faces of the same object under study. This inter-disciplinary increases the necessary amount of knowledges that we have to integrate in order to understand the dynamic of complex systems. Moreover, discipline such as ES encounter major difficulties when they want to perform experiments on natural systems. Indeed, the large spatial and temporal scales for instance, prohibit numerous replications of experiments. To avoid these difficulties, ES uses computer based models in order to perform virtual experiments. Until now, the majority of models are developed using programming languages and/or using simulation softwares that do not explicitly propose the tools for the integration of heterogeneous models. The increasing complexity in systems modelling and simulation (M&S) needs reliable softwares. Thus, there is a need for complete software environments dedicated to multi-disciplinary modelling and simulation (M&S). Some tools exist such as Modelica [1] or AToM3 [2]. The former is oriented towards the industry and the latter towards the research in M&S. The aim of our works is in between, i.e. between applied and theoretical M&S. It is to develop a computational environment dedicated to the researches in the ES. We comment this point at the beginning of the discussion.

Heterogeneous knowledges are supported by heterogeneous languages or formalisms. M&S is the keystone that articulate the representations shared between different disciplines. It provides numerous formalisms and computational methods,

* Corresponding author. Tel.: +33 5 61 28 54 36.
E-mail addresses: gauthier.quesnel@toulouse.inra.fr (G. Quesnel), raphael.duboz@cirad.fr (R. Duboz), ramat@lil.univ-littoral.fr (É. Ramat).

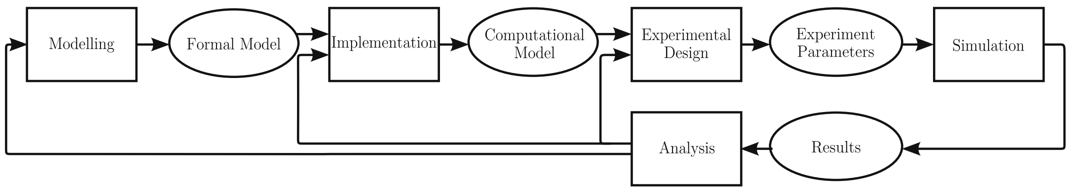


Fig. 1. The modelling and simulation cycle: boxes are actions and ellipses are the products of actions.

each one being adapted for a particular class of system. Moreover, it gives us the formal and operational tools to articulate (we can say to couple or to integrate) heterogeneous models. Some works refer to the modelling of heterogeneous systems as Computer Automated Multi-Paradigm Modelling (CAMPaM) [3,4], multi-modelling [5,6] or integrated multi-modelling [7]. CAMPaM aims to simplify the modelling of complex systems by meta-modelling (the modelling of formalisms), model abstraction (relationship between models at different levels of abstraction) and multi-formalism (coupling between models describe in different formalisms). In this paper, we are in concern with the multi-formalism dimension of CAMPaM. We can address multi-formalism following three orthogonal directions:

- *The integration of several formalisms in a new one:* We can cite Vangheluwe [8] who introduced the DEA (Differential Algebraic Equations), Zeigler [9] with the DEV&DESS and Barros [10] 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 multi-formalism M&S. That is why we have based the development of our tool upon the operational and formal framework defined by Zeigler [9] and Kim and Kim [11]. These frameworks address both the technical and the theoretical issues of multi-formalism.

Another important aspect regarding the M&S tools is the possibility to deal with all the phases of the classical M&S cycle (Fig. 1).

In this cycle, the modeller begins by designing the model. This model can be specified using one or several formalisms. Secondly, the modeller implements the model. Thirdly, the modeller defines an experimental frame (value of the parameters, observation methods, number of replicas, etc.). Finally, the results are analyzed. This is a cycle since we can reiterate actions after the analysis by modifying either the experimental frame, the implemented model or the formal model.

In this paper, we introduce a new framework for the multi-modelling, simulation and analysis of complex systems, the Virtual Laboratory Environment. We first give an overview of VLE recalling the operational and formal background of our works and the general architecture of the framework. Thereafter, we present the tool in the context of the M&S cycle. Then, we give an example of a multi-model implemented using VLE. We close this paper by a discussion and some opened works.

2. Overview of the Virtual Laboratory Environment (VLE)

2.1. Formal and operational background

Our works take place in the context of th M&S theory defined by Zeigler [12]. M&S theory tends to be as general as possible. It addresses 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.

Following Vangheluwe [8], we think that the Discrete Event System Specification formalisms (DEVS) is a good candidate to be a common denominator for multi-modelling. Indeed, it has been shown that classical or timed automata, discrete time systems, Petri net and state charts for instance, can be considered as DEVS sub-formalism [9]. Even, continuous systems can be approximated using DEVS integrators. Many formal extensions to DEVS were carried out, therefore, we advise the second edition of Zeigler's book [9] to have an overall picture of these works.

DEVS defines an atomic model as a set of input and output ports and a set of state transition functions. Every atomic model can be coupled with one or several other atomic model in order to build a coupled model. This operation can be repeated to form a hierarchy of coupled models. The set of atomic and coupled models and their connections is named the structure of the model. One important point is that DEVS is an operational formalism, i.e. it provides the algorithms (the abstract simulators) that implement the formal models. VLE is based on an extension of DEVS, the Dynamic Structure Discrete Event formalism (DSDE) [13] which provides the abstract simulators for:

- Parallel DEVS (PDEVS) [9] for the parallelization of atomic models.

- Dynamic Structure DEVS (DSDEVS) [14] for the M&S of systems where drastic changes of structures and behaviors can occurred over time.

The implementation of the DSDE abstract simulators gives to VLE the ability to simulate distributed models and to load and/or delete atomic and coupled models at runtime.

VLE proposes several simulators for particular formalisms. In addition to DSDE, VLE implements three others DEVS extensions:

- The Quantified State System (QSS) [15] for the resolution of differential equation systems with a quantification method.
- The Cellular Automata DEVS (CellDEVS) [16] for the M&S of spatialized systems.
- The CellQSS specification [17], the association of CellDEVS and QSS extensions for the resolution of spatialized differential equation systems.

Moreover, VLE owns DEVS simulators for:

- The integration of ordinary differential equations using the fourth-order Runge–Kutta method.
- The implementation of a discrete time method for the resolution of difference equations.
- The simulation of Petri nets.

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. These algorithms can be found in [9]. As it is not always possible to adopt a common formalism for all sub-models of a system, we have implemented the DEVS-Bus introduced by Kim [11]. DEVS-Bus enables interoperating with diverse discrete event or discrete time modelling formalisms. Any formalism we want to integrate can be wrapped in a DEVS abstract simulator [18], this is the co-simulation approach.

2.2. General architecture

As we have said in the previous section, VLE is oriented toward the integration of heterogeneous formalisms. Furthermore, VLE is able to integrate specific models developed in most popular programming languages into one single multi-model. To achieve that, VLE provides complete libraries named *VLE Foundation Libraries* (VFL) and tools for models design and simulation. In the following, we introduce the VLE architecture and Application Programming Interface (Fig. 2).

There are five applications in VLE environment based on the VFL, each one using a set of particular components (i.e. plug-ins):

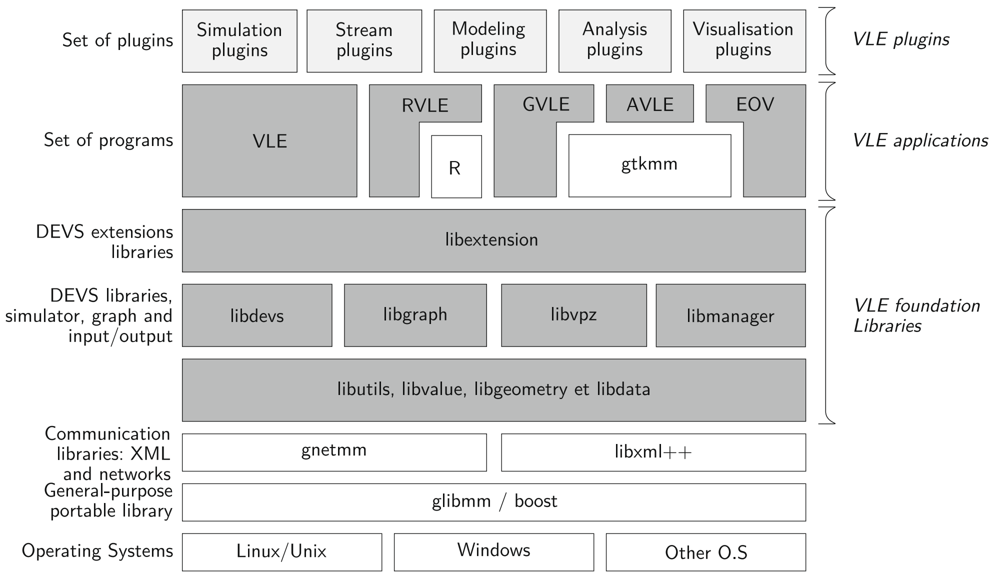


Fig. 2. Representation of the VLE framework Application Programming Interface (API). Dark grey boxes are the VFL and the applications that we have developed. Clear grey boxes are plug-in or components developed by users to extends VLE API (simulations plug-in, etc.) and white boxes are external libraries coming from the open sources projects to increase the portability (*glibmm*, *boost*, etc.) or to extends VLE (*R project* [19]).

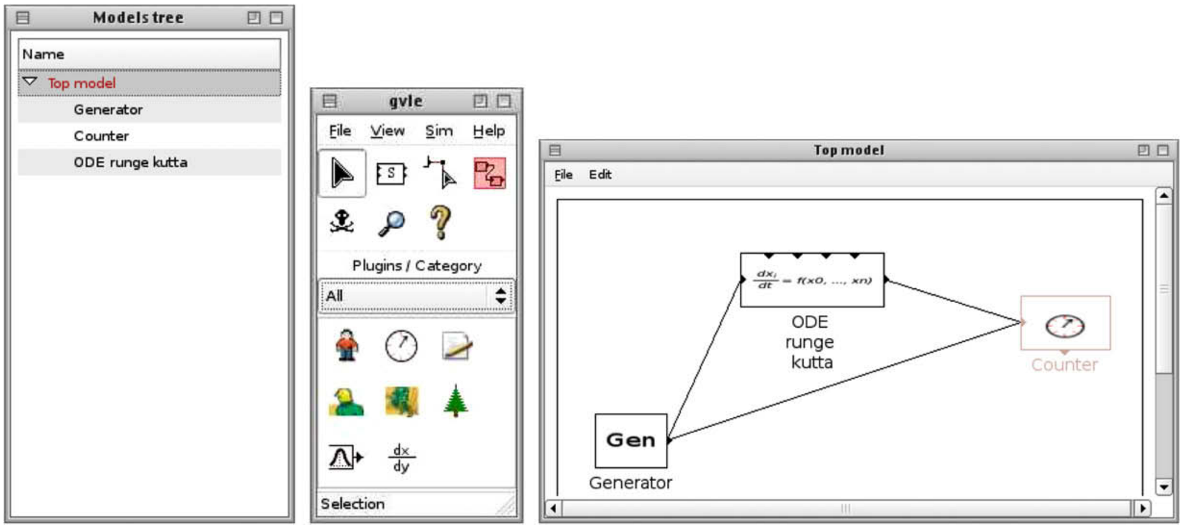


Fig. 3. GVLE is used for the modelling and the experimental design. It provides graphical user interfaces to manipulate atomic and coupled models. Icons of the center window represent tools for models management and simulation plug-in. The window on the right is a view of coupled models and the left window shows the hierarchy of coupled models.

- GVLE is a graphical user interface. It provides tools to visually construct a hierarchy of coupled models. A modelling plug-in can be use to define and to modify the behaviour of atomic models displaying a text editor where DEVS functions can be coded. Moreover, GVLE enables the definition of experimental frames.
- EOV, *the Eyes Of VLE*, is a graphical application which displays the values of states during simulation. EOV is a set of visualization plug-ins. A particular plug-in defines the type of visualization like colored grided surfaces or curves for instance.
- VLE is the core of the environment. The four 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 in the previous section. To perform simulations, VLE records the experimental frame generated by GVLE and then dynamically loads simulation and visualisation components of EOV and finally connect them to the DEVS-Bus. The 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 [19].
- RVLE (R for VLE) is a R-Package to build experimental frames, to launch the simulation and to get the results of the simulation within the R environment.

The VLE framework is written in the standardized C++ programming language [20]. C++ ensures the compatibility with a large number of operating systems and the interoperability with the major programming languages as Java, Fortran or Python for instance. We increase the portability of VLE using the portable libraries provided by the GNU Project [21]. The choices of C++, the GNU libraries and the concepts of components have made the VLE an efficient and portable environment, easily modifiable and fast to develop.

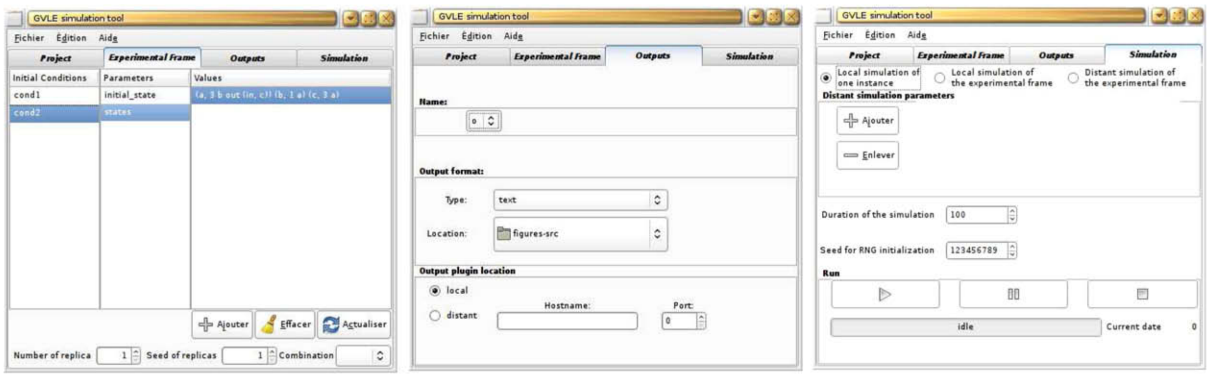
3. VLE and the modelling and simulation cycle

In this section, we describe the M&S activity using VLE. We refer to the M&S cycle shown by Fig. 1. This way, we illustrate how the modeller uses VLE and realizes the complete cycle. The sub-sections correspond to an action (boxes) of the figure. We do not present the implementation phase of the cycle because it is model dependent and mainly an hand coding task. It corresponds to the implementation of the functional interface presented in Section 3.3.

3.1. Modelling

GVLE, the Graphical User Interface for VLE, sustains the modelling activity. GVLE modelling 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 in order to build coupled models. Atomic and coupled models are represented by black boxes.[1] Doing that, the modeller build a hierarchy of coupled models. The whole hierarchy is shown on the left window as illustrated by Fig. 3.

[1] The DEVS property of closure under coupling ensures that a coupled model is equivalent to an atomic one [9].



Initial Conditions	Parameters	Values
cond1	initial_state	(a, 3 b out (in, c)) (b, 1 a) (c, 3 a)
cond2	states	

Number of replica 1 Seed of replicas 1 Combination

Name: 0

Output format:

Type: text

Location: figures-src

Output plugin location

[✓ local] distant Hostname: Port: 0

[✓ Local simulation of one instance] Local simulation of the experimental frame Distant simulation of the experimental frame

Distant simulation parameters

Duration of the simulation 100

Seed for RNG initialization 123456789

Run

idle Current date 0

Fig. 4. Definition of experimental frame. The window on the left is used to set the initial conditions. In the center window, the modeller defines the state variables to observe and the visualization component to use. The right window defines the duration of the simulation. It controls the simulation by using the run, pause and stop actions.

The modelling components, or plug-ins, own informations about the dynamics of models they instantiate. They define which simulation components must be used and some other features specific to particular simulation components. For instance, the ordinary differential equation component [18] of Fig. 3 specifies the number and the form of the equations to solve. In addition to the pre-defined modelling components, the modeller can use a generic component which represents a generic atomic DEVS model. The modeller uses this component to define a particular dynamic embedded in DEVS.

Results of the modelling activity (i.e. the definition of the structure and the dynamics of models) are stored in a particular XML format, the VPZ (*Virtual laboratory Project Zip*) XML application file.[2]

3.2. Experimental design

In addition to the structure and the dynamic of models, the VPZ format stores the experimental frame. In the experimental design phase, the modeller chooses the states to observe and how to look at their evolution over time, i.e. the visualization component to use. Moreover, the modeller defines the initial conditions and the duration of the simulation. The modeller can define the variation domain for the initial parameters. Thereafter, VLE computes the number of simulations needed to achieve the experimental plan. The default behavior is an exhaustive experimental plan (i.e. the cross-product of initials conditions) (see Fig. 4).

In VLE, we have defined two particular types of port in addition to input and output ports in DEVS models: the initialization and state ports. Initialization ports receive initial values of parameters. State port are connected to one or several measure objects which receive the state values under observation. Then, the measure object sent values to a specialized component. Fig. 5 illustrates the use of state ports in VLE. The values of observed states can be treated by visualization components or storage components.

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. We call them "discrete measures".
- When an internal or external transition occurs, the associated simulator receives measure events at the same time. The measure event is treated just after the transition event. We call them "event measures".

Event measures do not modify model states. Such modifications could occurred for discrete measures. To avoid state changes when an observation occurs, the function managing discrete measures has the following prototype:

```
Value observation(const ObservationEvent& event) const;
```

The C++ keyword *const* at the end of the definition indicates that it is not possible to change the value of the members of the class. This assert that the modeller cannot change the states of models by using this function.

The RVLE application presented in Section 2.2 can be used for the design of experimental frames. Indeed, it is possible to generate experiment plans by using specialized R packages (the classical methods for linear sensitivity analysis for instance). RVLE translates the generated set of parameters values into the VPZ format, runs the simulations and returns the results into R objects.

Once the modelling and experimental design activities are ended, the VPZ file contains all the necessary informations that the VLE core program needs to perform simulations. The modeller can reproduce exactly the same experimental frame by

[2] http://vle.univ-littoral.fr/vle-0.5.0.dtd.

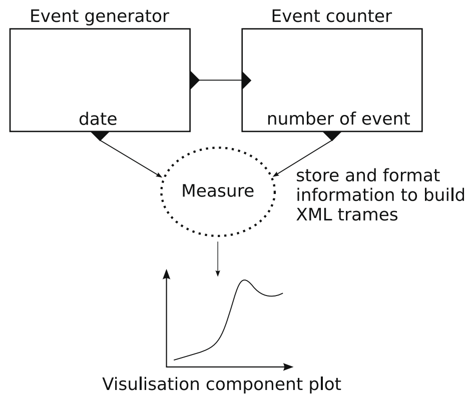

Event generator
Event counter
date
number of event
Measure
store and format information to build XML trames
Visulisation component plot

Fig. 5. The diagram presents an example of the management of output within 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. This object sends values to the visualization component using network or memory transfer.

reusing an existing VPZ file, even for the stochastic models. Indeed, the seed of the pseudo-random number generator is recorded in the VPZ file.

3.3. Simulation

The VLE core program performs the simulations. It first parses the VPZ file. Doing that, it gets all the necessary informations in concern with the model structures, dynamics and experimental frame. Thereafter, the VLE core program loads the useful simulation and visualization components, and builds the coupled models network. Then, it sends an initialization message to every model and starts the simulation.

VLE implements the abstract simulators of the DSDE and the DEVS extensions. It uses the Parallel DEVS technique [9]. 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:

```
Time init();
void internalTransition(const Time& time);
void externalTransition(const ExternalEventList& events);
Time timeAdvance() const;
void output(const Time& time, EventList& out) const;
void finish();
```

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

```
type confluentTransitions(const Time& time, const ExternalEventList& e) const;
```

This function is called when events occur at the same time. It can be defined to choose the order of treatment for events between internal and external events. The two functions that realize the experimental frame are:

```
Dynamics(const InitEventList& events);
Value observation(const ObservationEvent& event) const;
```

The first one, the constructor of the class, 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.

3.4. Analysis

Analysis of model behaviors can be either qualitative or quantitative. In the first case, the modellers use visualization tools. In the second case, they use statistical tools. VLE does not own such tools but provides several format for data saving to be compatible with the most popular statistical tool. The simpler way is to directly store data in text files. VLE records one line per date and one column per observed state. Data can be saved using CSV, R format or the Extended Markup Language format for Statistical Data (StatDataML). Thus, the user can directly load the simulation results in a statistical software such as R, Splus, MATLAB or Octave for instance. Moreover, VLE proposes the RVLE (see Section 3.1) to embed VLE within R. Analysis can be performed from this tool, well known for its numerous statistical methods. The AVLE application (see Section 3.1) is a GUI to make simple analysis of data using the R libraries. It is able to read VPZ files generated by the GVLE application, and to generate graphics considering the VPZ contents.

For the qualitative analysis, VLE provides the EOV application. EOV owns the visualization and storing components. The visualisation components draw the evolution of the states of models at runtime. They are developed using inheritance of the specific EOV class. Nowadays, the VLE framework proposes the following visualisation components:

- 2D gridded data.
- 2D or 3D continuous spatialized data.
- Curves, histograms.
- Gauges to visualize values between a minimum and a maximum.

In the next section, we illustrate the effectiveness of the VLE environment to address multi-modelling by considering an agent model of firemen fighting against fire spread triggered by pyromaniacs.

4. Example

In this section, we illustrate the multi-modelling process within the VLE environment with an example. We model firemen fighting against the fire spread initiated by pyromaniacs. This model shows how VLE is used to integrate several formalisms and how it can addresses issues related to complex systems M&S. In this example, we couple spatialized ordinary differential equations embedded in a cellular automaton [17] and simple agents specified with the dynamic structure DEVS formalism [22].

4.1. Modelling

Two types of agent interact with the forest, the pyromaniacs and the firemen. For simplicity here, these two types of agents perform random walks.

4.1.1. Forest model

The fire spreading is modelled using the Cell-QSS extension [17] which merge the behaviour of a cellular automaton (Cell-DEVS, [16]) and an ordinary differential equations integrator (QSS, [23]). The forest model is a cellular automaton. Each cell represents a part of the forest and is characterized by the biomass σ_v of the combustible, and the average temperature T. We consider the following differential system [24]:

$$\begin{cases} \frac{\partial T}{\partial t} = -k(T - T_a) + K\Delta T - \mathcal{Q}\frac{\partial \sigma_v}{\partial t} \\ \text{with :} \\ \frac{\partial \sigma_v}{\partial t} = 0 \text{ for the inert cells} \\ \frac{\partial \sigma_v}{\partial t} = -\alpha\sigma_v \text{ for cells in fire} \end{cases}$$

where T is the average temperature of the cell; T_a the ambient temperature; k the factor between ambient temperature and T; K the diffusion factor to the neighbourhood of the cell; ΔT the laplacian of T; $\mathcal{Q}$ the characteristics factor of propensity to spread the neighbourhood of the cells; σ_v the mass of remaining combustible. The Cell-QSS extension model spatialized ODE and we use it to control the fire propagation; α is the combustion speed.

An inert cell will switch to state "fire" if it reaches a temperature called *ignition* T_{ig}. The cell switches to an inert state if its temperature is lower than T_{ig}.

We approximate the laplacian (i.e. the sum of the second derivative in comparison with the neighbourhood space). Thus, it is possible to specify the equation as a Cell-DEVS model (see [17] for a complete description).

4.1.2. Fireman and pyromaniac models

The "agent" model (fireman and pyromaniac) is a finite state automata (see Fig. 6). It is connected to the "move" model by its input port X_{moved} and output port Y_{move}. This model have two ports connected to the Cell-QSS model (a cell), X_{in} an Y_{out} (respectively, the input and output ports). The behavior of the agent model is a simple random walk. For each elementary displacement, the agent uses the state "`TEST`" to check if it has to perturb the cell.

4.1.3. Move model

The "move" model, is an executive DSDEVS model. It modifies the DEVS graph of the coupled model, (i.e. the connections between the cellular automaton and the agents). The behaviour of this model is a simple automaton. It waits for the request

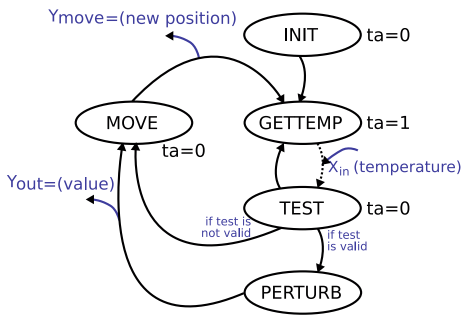


Fig. 6. This state diagram of the agent model shows two behaviours. The first one is the random walk with the `MOVE` state and the second one is the test of the temperature of the cell (states `GETTEMP`, `TEST` and `PERTURB`) its eventual perturbation.

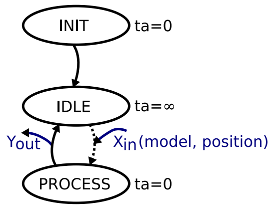


Fig. 7. State diagram of the "move" model. The ellipses represents the states, lines are internal transitions and dot-lines are external transitions. The `IDLE` state is an infinity state while no event arrived on X_{in} port. The internal transition between `PROCESS` and `IDLE` is used to modify the connections and send the acknowledgment.

of an agent. When an event arrives on port X_{in}, the "move" model add a new connection (based on the associated event position value), destroys the old one, and finally sends an acknowledgment event to the model. This behaviour is illustrated by the state diagram on Fig. 7.

Fig. 8 shows the structure of the graph of coupled models. Here, the agent is coupled with the cell "Forest12". The "move" model is able to change connections to simulate the displacement of the agent on the cellular automaton.

4.2. Implementation

The implementation task in the cycle of modelling and simulation (see Fig. 1), is the writing of a source code starting from DEVS formal specifications. In VLE, the implementation of a DEVS model is achieved by an inheritance of the DEVS atomic model class, a DS-DEVS executive model class or another DEVS extension of the VLE framework. We present here the implementation of the models previously defined.

4.2.1. Forest

The forest is a Cell-QSS model. Cell-QSS is an extension in the VLE, i.e. a model (atomic or coupled) with a specific functionality. It is possible to use it without knowledge of DEVS. For instance, to implement an ODE system in VLE, the model class inherits from the QSS class and the modeller just defines the `qss::compute` abstract function to specify the equation. The modeller does not use the functions defining the DEVS interface (i.e. ta, δ_{int}, δ_{ext}, λ and δ_{con}). Nevertheless, if the modeller wants to modify the behaviour of QSS model, he can override these abstract functions inherited from the DEVS extension.

The Cell-QSS class works in the same way and adds some functionalities to the QSS class. It provides an API to check the neighbourhood state values, the temperature in our example, to update its own states.

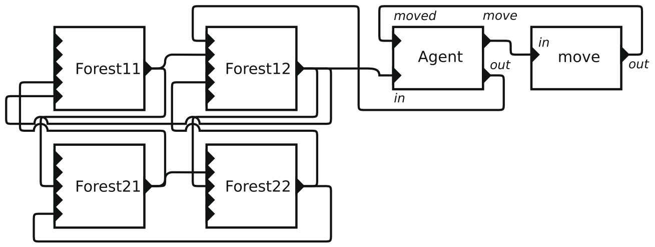


Fig. 8. This graph illustrates the coupling of four cells of the cellular automaton with the agent model and the move model.

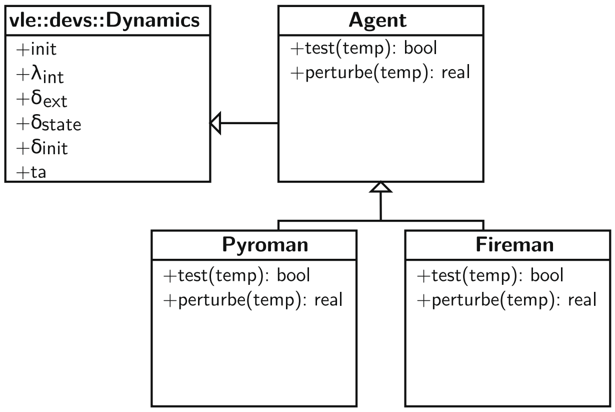


Fig. 9. Inheritance tree of the VLE DEVS atomic model `devs::Dynamics` and the abstract `Agent` class. Only the `Fireman` and `Pyromaniac` classes are used by the simulator.

4.2.2. Agent model

The agent is a generic class. Pyromaniac and fireman classes inherits from the agent class for the displacement behavior (see Fig. 9), and add specificities for the management of the cellular automata perturbations. The agent class provides an `Agent::perturb` function which sends a new temperature to the cell and a `Agent::test` function to check if the `Agent::perturb` is allowed to change the temperature of the cell.

4.2.3. Move model

The "move" model controls the connections of the agent. When it receives an order of modification, it builds the new connections and destroy the old ones. This model inherits from the `devs::Executive` class, in order to be able of manipulate the connection graph. The important operation is the internal transition function when connection changes occur. As the "move" class inherits from the `devs::Executive` class, it can use a specific API which allows to build, remove, or change connections.

4.3. Simulation

The specification of the experimental frame is given by the VPZ file. It describes all the set up of the system: the graph of coupled models, how models are initialized and how to observe them. Nevertheless, when the number of model is high, writing this file by the hand or with the GVLE program is potentially a long and hard task. In order to simplify the construction of this file, we use a translator which automatically build a graph of models and connections and theirs initialization thanks to a specific input.

In the following example, we use the VLE `MatrixTranslator` which allows the automatic generation of lattice models (where each cell is an atomic or coupled DEVS model and connections depending on the type of neighbourhood, Moore or Von Neuman). The following listing is the VPZ syntax to use to specify a Cell-DEVS model:

```
<celldevs>
  <grid>
    <dim axe = "0" number = "20"/>
    <dim axe = "1" number = "20"/>
  </grid>
  <cells connectivity = "moore" library = "firemanqsscell">
    <prefix>c</prefix>
  </cells>
  <agents>
    <agent name = "pyromaniac_1" classname = "pyromaniac">
      <parameter x = "4" y = "17" value = "1.0"/>
    </agent>
    <agent name = "fireman_1" classname = "fireman">
      <parameter x = "3" y = "8" value = "−1.0"/>
    </agent>
  </agents>
</celldevs>
```

Fig. 10 illustrates the use of the EOV application to display the evolution of the fire spread.

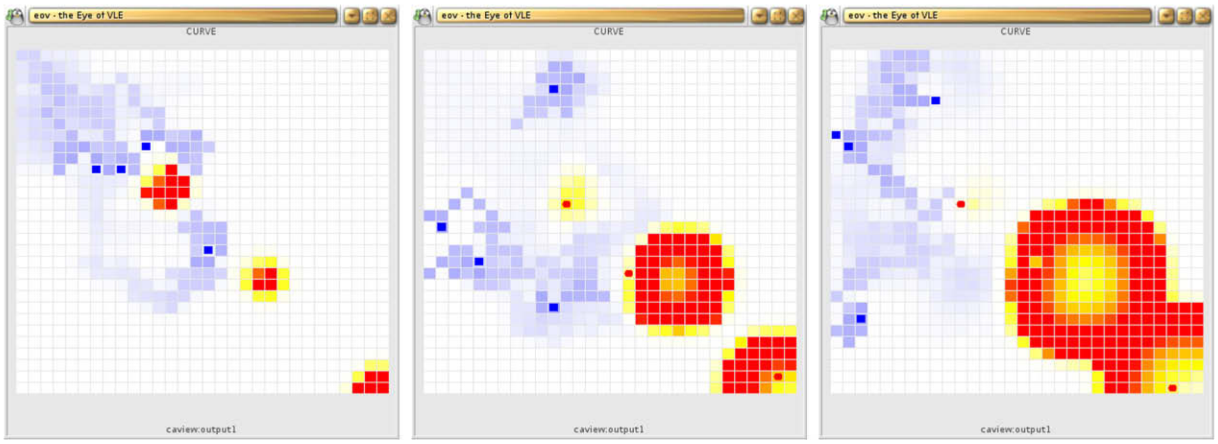


Fig. 10. Screenshots from the EOV program. Firemen (small blue box) and pyromaniac (small red circles) are located on the forest (the grid). White cells have an average temperature, blue cells reflect cold temperatures and red to yellow display cells in fire with high temperatures. Time advances from picture of the left to picture of the right. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5. Discussion

The discussion is divided in three parts. In order to better situated our work, the first part recalls the context of development of VLE and the second part compares it with similar projects and tools. In the last part, we discuss the characteristics and originalities of our environment.

5.1. Context of development

Our works have started with multi-disciplinary issues emerging from the interaction between computer scientists and biologists [25]. Considering these works, we think that the integration of heterogeneous models and the respect of the M&S cycle are the key issues to provide a complete and reliable software environment for natural complex systems modelling. So, VLE has evolved toward a complete multi-modelling and simulation environment [26].

As we have said in Section 1, VLE is now oriented toward the Environmental Sciences (ES). It is used in many projects of major French research institutes[3] and funding by the French National Research Agency.[4] For instance, in the CHALOUPE project, VLE supports a model for the simulation of the dynamic of exploited marine biodiversity and the viability of fisheries. In the REMIGE project, VLE models the impact of climate changes on marine top predators. In the Record project, VLE is use to simulate agronomics and decisional models using sensitivity analysis and simulation based optimisation. Recent papers in concern with VLE have been published in international conferences [27,28,18]. These works have guided the development of VLE which had to fit with functional needs and objectives of very different multi-modelling applications. To be relevant, VLE has to be built at the same time on a strong theoretical basis and to offer the whole of the necessary tools for practitioners. That is why we said in Section 1 that our works is between theory and application.

5.2. Similar projects and tools

5.2.1. DEVS based tools

There is a large number of DEVS based simulators. A rigorous comparison of the performances or the implementation of simulators is very difficult, near impossible. One possibility is to choose a test model and implement it using the different tools. It is a very time consuming task. Moreover, the choice of the test model is not trivial.

We have consider some platforms based on the DEVS formalism with VLE. We have not compared the whole of platform's characteristics rather than focus on the main features of VLE: i.e. the implementation of major DEVS extensions, the experimental frames, the ability to perform distributed simulation, the separation between the dynamic of atomic models and theirs structure. We have considered the list provided by G. Wainer. References to platforms (ADEVS, CD++, DEVS/C++, DEVS/Java, JDEVS, etc.) can be found at http://www.sce.carleton.ca/faculty/wainer/standard/tools.htm.

Some of these platforms are specialized by modelling domain. We can cite CD++ [16] for the modelling of cellular automata models. CD++ implements specialized algorithms for optimization, parallelization and distribution but can be used to develop classic atomic DEVS model too. Another example is DEVS/Java, a simulator written in Java that illustrates the coupling of DEVS and HLA for distributed simulation. It permits now to model variable structures, differential equation systems and cellular spaces. Some others are particularly well adapted for DEVS modelling (DEVS/C++ for instance) but do not imple-

[3] INRA – http://www.inra.fr, Cirad – http://www.cirad.fr, Ifremer – http://www.ifremer.fr, IRD – http://www.ird.fr.
[4] ANR – http://www.agence-nationale-recherche.fr/.

ment experimental frames. ADEVS, a recent C++ library heavily based on generics programming to build efficient simulation but losing an easy way to reuse models.

We discuss the choice to develop a new DEVS kernel in the Section 5.3.

5.2.2. Multi-modelling environments

In the following, as far as we know, we present the four major multi-modelling environments:

ATоM3 (A Tool for Multi-formalism and Meta-modelling) [29] is a modelling project lead by Vangheluwe at McGill University. Contributions fall in the field of meta-modelling which focus mainly on formalism modelling and on transitions between formalisms.

Rube [30] is a software of modelling and visualization framework. It is based on OOPM, an object-oriented approach for the modelling of static or dynamic physical systems. The models are structured in hierarchies of homogeneous or heterogeneous models [31].

Modelica is an object-oriented language for modelling engineering systems [1]. It results from an important international research group on multi-formalism modelling and simulation. Modelica allows the simulation of heterogeneous and complex systems provided by a numerous libraries. Modelica translates models into a unique sophisticated symbolic techniques to perform coupling.

Ptolemy II [32], is a framework of modelling and simulation based on the concept of components. It is also an object-oriented approach of modelling where the focus is on the assembly of concurrent components. Ptolemy II includes a suite of domains (continuous times, discrete-event, finite state machines, etc.) and a component library. Most components are domain polymorphic (i.e. they can operate in several domains). Ptolemy II uses the co-simulation approach for the coupling of domains.

We discuss the position of VLE regarding these tools in the following section.

5.3. Characteristics of VLE

The frameworks presented above (in Sections 5.2.1 and 5.2.2) are originals and can be use to model, simulate, analysis or visualize dynamics of complex systems. Nevertheless, they could not have been chosen to serve our needs. Indeed, several features dispersed between these tools had to be present in the same tool:

- Multi-modelling abilities.
- A general formal basis for modelling dynamic systems and an associated operational semantic.
- A modular and hierarchical representation of the structure of coupled models with associated coupling and coordination algorithms.
- Coupling of pre-existing models.
- Distributed simulations.
- A component based development for the acceptance of new visualization tools, storage formats and experimental frame design tools.
- Free and open source software.

VLE proposes all these features in a unique environment. Rather than base our framework on a language (like Modelica), or a development methodology (like Rube or Ptolemy), we have chosen a formal basis which is general enough to potentially integrate new formalisms in the future [9]. Some works bring together software communities. For example, Barros uses the OOPM and DSDEVS [33], or D'Abreu model continuous systems using Modelica and DEVS [34]. To our opinion, these works illustrate that DEVS is well adapted as a common formalism for multi-modelling, as propose by Vangheluwe [8]. Many formalism with explicit time can be embedded in DEVS. Moreover, formalisms where time is not explicit, like classical Petri net, finite state automata or state chart, can be specified using DEVS. Simulators of models that are not well suited to be specified in DEVS can be wrapped to the abstract simulators through the DEVS functions. In the same way, pre-existing models can be integrated in DEVS simulators.

Another important feature of VLE is its component based development. The models, the simulator and peripheral tools are organized in libraries. Components can be coupled together simply by specifying the structure of coupled models and the experimental frame in an XML application (the VPZ format). The separation of models from their behaviors may reduce the modelling complexity of real-world systems. It allows designers to easily construct candidate models with different structures while using the same components [9].

The VPZ format gives VLE the ability to easily communicate with others applications. By parsing the VPZ file, and with the atomic model specifications, any other simulation tool is able to build and simulate the specified model, under the same experimental conditions. Then models in VLE are open to comparative verifications. This is a fundamental issue in complex scientific M&S activities. Moreover, the development of bridges between VLE and other modelling software open our framework to a wide community.

Finally, the VFL (VLE Foundation Libraries) are easily reusable by another software. For instance, RVLE, the VLE R package expands VLE possibilities (R provided very wide range of statistical and graphical tools) and the R users can keep their working environment and use VLE like another functions of the R language. Theses R functions, connected to the VFL, allow to load VPZ files, modify the experimental frames, and start simulation. These simulation can be on the local host or on a grid computing. For demonstration, we also provide a binding with the same functionality for the Python programming language.

6. Conclusion and perspectives

In this paper, we have presented the VLE framework. It is a complete and powerful DEVS environment for modelling, simulation and analysis of complex systems. We have mainly developed this environment to build a complete software that enables the complete realization of the modelling cycle (Fig. 1). We have shown how the general architecture of VLE can support this cycle for multi-modelling through an example where a cellular automata, spatialized ordinary differential equations and a finite state automata are coupled 4.

Complex systems are better addressed considering the multi-modelling as a basis for there construction. Nowadays, the models are mainly developed using programming languages or simulation softwares that do not explicitly propose the tools for the integration of heterogeneous models. As VLE is based on the M&S theory defined by Zeigler [9], it offers these tools to couple heterogeneous models and to simulate them relying on a formal basis. Moreover, the separation of the structures of models from their behaviours improve modularity and reuse of models. These are very important features considering scientific use of M&S:

- The formal basis ensures that the coupled model is correctly simulated regarding the time advance and the dynamic of states.
- The specification of model structures and behaviours in a open format improve the communication and the verification of models.
- The model library increases development speed and reusability.
- The validation process is improved by the common observation method of DEVS models.

To summarize, we think that the use of VLE, can reduce the development time, increase the quality of models and simplify the necessary efforts for coupling heterogeneous models. It is very impacting on a research field such as Environmental Sciences for which we are developing models [27,28]. In addition, we think that VLE is general enough to support other domains. Indeed, VLE enables continuous and/or discrete time models, spatial or not.

The concept of *Translator* we have briefly introduced in this paper seems very promising. These works are close to those in concern with model processing and meta-modelling (see De Lara and Vangheluwe [29] for instance). Some of our recent works on the DEVS specification of Multi-Agent Systems use the concept of *Translator* [22,35]. It is a powerful tool to facilitate the development of domain-dependant applications. We are currently using the meta-modelling concept for simulation based optimization and the modelling of decision in cropping systems.

Informations concerning VLE, including sources, examples, models and documentations are available on the VLE website.[5]

Acknowledgements

We would like to thank the projects CHALOUPE and REMIGE of the French Research National Agency (ANR) and the RECORD project from INRA to have supported this work.

References

[1] M.M. Tiller, Introduction to Physical Modeling with Modelica, Kluwer Academic, 2001.
[2] J. de Lara, H. Vangheluwe, AToM^3: a tool for multi-formalism and meta-modelling, Lecture Notes in Computer Science 23 (6) (2002) 174–188.
[3] H. Vangheluwe, J. de Lara, P.J. Mosterman, An introduction to multi-paradigm modelling and simulation, in: F.J. Barros, N. Giambiasi (Eds.), Proceedings of the AIS 2002—Simulation and Planning in High Autonomy Systems, Society for Modelling and Simulation International, Lisbon, Portugal, 2002, pp. 9–20.
[4] H. Vangheluwe, J. de Lara, Computer automated multi-paradigm modelling: meta-modelling and graph transformation, in: Proceedings of the 2003 Winter Simulation Conference, vol. 1, 2003, pp. 595–603.
[5] P.A. Fishwick, B.P. Zeigler, A multi-model methodology for qualitative model engineering, ACM Transaction on Modeling and Simulation 2 (1) (1992) 52–81.
[6] P.A. Fishwick, Simulation Model Design and Execution, Prentice Hall, 1995.
[7] P.A. Fishwick, Toward an integrative multimodeling interface: a human–computer interface approach to interrelating model structures, Simulation 80 (9) (2004) 421–432.
[8] H. Vangheluwe, DEVS as a common denominator for multi-formalism hybrid systems modelling, in: A. Varga (Ed.), IEEE International Symposium on Computer-Aided Control System Design, IEEE Computer Society Press, Anchorage, AK, 2000, pp. 129–134.
[9] B.P. Zeigler, T.G. Kim, H. Praehofer, Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Academic Press, 2000.

[5] http://vle.univ-littoral.fr.

[10] F.J. Barros, Dynamic structure multiparadigm modeling and simulation, ACM Transactions on Modeling and Computer Simulation 13 (3) (2003) 259–275.
[11] Y.J. Kim, T.G. Kim, A heterogeneous simulation framework based on the DEVS BUS and the high level architecture, in: Winter Simulation Conference, Washington, USA, 1998, pp. 421–428.
[12] B.P. Zeigler, Theory of Modeling and Simulation, Wiley–Interscience, 1976.
[13] F.J. Barros, Modeling formalisms for dynamic structure systems, ACM Transactions on Modeling and Computer Simulation (TOMACS) 7 (1997) 501–515.
[14] F.J. Barros, Dynamic structure discrete event system specification: formalism abstract simulators and applications, Winter Simulation 13 (1) (1996) 35–46.
[15] E. Kofman, A second order approximation for devs simulation of continuous systems, Journal of the Society for Computer Simulation International 78 (2).
[16] G.A. Wainer, N. Giambiasi, Application of the Cell-DEVS paradigm for cell spaces modelling and simulation, in: Simulation, vol. 76, 2001, pp. 22–39.
[17] D. Versmisse, E. Ramat, Management of perturbations within a spatialized differential equations system, in: European Simulation and Modelling Conference, SCS, Porto, Portugal, 2005, pp. 520–524.
[18] G. Quesnel, R. Duboz, E. Ramat, DEVS wrapping: a study case, in: Proceedings of the Conference on Conceptual Modeling and Simulation, Genoa, Italy, 2004, pp. 374–382.
[19] 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.
[20] B. Stroustrup, The C++ Programming Language, Addison Wesley, 1986.
[21] Free Software Foundation, GNU Operating System: GNU's Not Unix, 1984. <http://www.gnu.org>.
[22] G. Quesnel, Approche formelle et opérationnelle de la multi-modélisation et de la simulation des systèmes complexes – apports pour les systèmes multi-agents, Ph.D. Thesis, Université du Littoral Côte d'Opale, 2006.
[23] E. Kofman, S. Junco, Quantized state systems: a DEVS approach for continuous systems simulation, in: Transactions of the Society for Computer Simulation International, vol. 18, Society for Computer Simulation International, San Diego, CA, USA, 2001, pp. 123–132.
[24] A. Muzy, G. Wainer, E. Innocenti, A.A.J.-F. Santucci, Comparing simulation methods for fire spreading across a fuel bed, in: F.J. Barros, N. Giambiasi (Eds.), Proceedings of the AIS 2002—Simulation and Planning in High Autonomy Systems, Lisbon, Portugal, 2002, pp. 219–224.
[25] E. Ramat, P. Preux, Virtual Laboratory Environment (VLE): a software environment oriented agent and object for modeling and simulation of complex systems, Simulation Modelling Practice and Theory 11 (2003) 45–55.
[26] G. Quesnel, R. Duboz, E. Ramat, M.K. Traoré, VLE – a multi-modeling and simulation environment, in: Moving Towards the Unified Simulation Approach, Proceedings of the Summer Simulation 2007 Conference, SCSC – ACM, San Diego, USA, 2007, pp. 367–374.
[27] D. Versmisse, C. Macher, E. Ramat, J.C. Soulié, O. Thebaud, Developing a bioeconomic simulation tool of fisheries dynamics: a case study, in: Proceedings of the International Congress on Modelling and Simulation, International Society for Computer Simulation, Christchurch, New Zealand, 2007, pp. 2799–2805.
[28] G. Quesnel, R. Duboz, D. Versmisse, E. Ramat, DEVS coupling of spatial and ordinary differential equations: VLE framework, in: Proceedings of the Open International Conference on Modeling & Simulation Conference, Clermont Ferrand, France, 2005, pp. 281–294.
[29] J. de Lara, H. Vangheluwe, AToM^3: a tool for multi-formalism and meta-modeling, Lecture Notes in Computer Science 23 (6) (2002) 174–188.
[30] J.F. Hopkins, P.A. Fishwick, The *rube* framework for personalized 3D software visualization, Lecture Notes in Computer Science 22 (69) (2002) 546–549.
[31] P.A. Fishwick, Extending object oriented design for physical modeling, Technical Report, 1996.
[32] E.A. Lee, Overview of the Ptolemy Project, Technical Memorandum No. UCB/ERL M03/25, 2003.
[33] F.J. Barros, B.P. Zeigler, P.A. Fishwick, Multimodels and dynamic structure models: an integration of DSDE/DEVS and oopm, ACM Transactions on Modeling and Computer Simulation (TOMACS) 7 (1997) 501–515.
[34] M.C. D'Abreu, G.A. Wainer, M/CD++: modeling continuous systems using Modelica and DEVS, in: Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005, pp. 229–236.
[35] R. Duboz, D. Versmisse, G. Quesnel, A. Muzzy, E. Ramat, Specification of dynamic structure discrete event multiagent systems, in: Agent Directed Simulation (Spring Simulation Multiconference), Huntsville, AL, USA, 2006.