A Package System for Maintaining Large Model Distributions in VLE Software

Gauthier Quesnel and Ronan Trépos
INRA, UR875 Biométrie et d'Intelligence Artificielle
F-31326 Castanet-Tolosan, France
Email: gauthier.quesnel@toulouse.inra.fr

***Abstract*—The Modeling and Simulation (M&S) is becoming a central activity in order to build, study and analyze new systems. To improve activities of M&S, we need to develop collaborative technologies. In this context, we develop the application software *Virtual Laboratory Environment* (VLE) to model, simulate and analyze dynamic systems. VLE is based on the *Discrete Event System specification* (DEVS) formalism, a widely recognized specification for modeling and simulating discrete events systems. The main features of the DEVS formalism are a modular and hierarchical approach of the M&S and a relative simplicity to develop the simulation algorithms. Researchers and engineers from different comunities used VLE to develop and study models. However, the modelers need to share source code in order to reuse, couple and combine models. It is not sufficient because they are not helped for maintenance and version upgrades issues. In this paper we present a package system manager that greatly helps modelers to publish source code, binary code, exchange models, data and software application in VLE.**

Packages, Package System Manager, Distribution, Discrete Event Modeling and Simulation.

I. Introduction

The Modeling and Simulation (M&S) is becoming a central activity in order to build, study and analyze new systems. M&S allow researchers, engineers and grant students to study the behavior of systems through virtual representation. Thus, M&S is increasingly used in industrial and research fields such as environment, agronomy, manufacturing, bio-medical systems. However, the cost of M&S activities increases with the complexity of models and the amount of communications between stakeholders (e.g., sub-domain experts, simulator users, simulator engineers, and final system users). To address the increasing costs of M&S activities, collaborative technologies must be introduced to support these activities.

Since 2003, we develop the M&S software environment called VLE [1]. VLE is based on the Discrete Event System Specification (DEVS) [2]. This formalism comes from the Theory of Modeling and Simulation (TMS) defined by Zeigler [2]. The TMS tends to be as general as possible. It addresses major issues of computer sciences from artificial intelligence to model design and distributed simulations. TMS aims to develop a formal and operational framework for the specification of dynamical systems. Indeed, it has been shown that classical or timed automaton, discrete time systems, Petri net and state charts can be considered as DEVS subformalisms. Even, continuous systems can be approximated using DEVS numerical solvers. Thus, the DEVS formalism can be candidate to be a common denominator for multi-modeling [3].

Today, VLE is used by many researchers, engineers, technicians in agronomic, environmental, transport and manufacturing research fields. They develop models, run simulations, analyze and optimize simulations. Modelers write many models with VLE. However, they have difficulties to use or reuse models or data from other users or from other research fields. In this paper, we propose to develop a package system to improve the sharing of models, data, source code and binary code between users of VLE. This project provides (i) packages to distribute models, data or source code (ii), tools to install, uninstall and build repositories and (iii) a workflow to update and maintain all the packages.

This paper is divided into five parts. In the first part, we explain the context of this work. In second part, we present the package, package system and workflow to build distribution. In third part, we show the first results of our work. We talk about the package systems and the returns of the users in the fifth part. Finally, we conclude this paper with conclusion and perspective.

II. CONTEXT

VLE is a modeling and simulation environment based on the DEVS formalism. VLE provides several software programs to develop atomic and coupled models networks, atomic models behavior, runs simulations and analyze results.

In order to build a collaborative environment for exchange or reuse of scientific knowledge between users of VLE, we have defined several tasks: (i) improve the usability of the DEVS formalism (ii) develop an integrated development environment (IDE), (iii) develop tutorials and training, (iv) develop small software programs that link VLE with major statistical and optimization tools.

A. DEVS implementation

The VLE software is an implementation of the DEVS formalism. VLE is set of software programs and software libraries developed in C++. The C++ was chosen because of (i) multi-paradigm (object-oriented programming and meta-programming) (ii) general-purpose programming language and (iii) compiled features. VLE implements the DSDE [4] abstract algorithms. DSDE allows building dynamic structure DEVS

1524-4547/12 $26.00 © 2012 IEEE
DOI 10.1109/WETICE.2012.34

models. As CD++ [5], ADEVS [6] or POWERDEVS [7], VLE provides C++ interfaces to develop atomic models. Thus, to develop atomic model, modelers develop C++ class that inherits this interface.

Listing 1: E.g API of the Dynamics class in VLE.

```
class Dynamics {
public:
virtual Time timeAdvance() const;
virtual void internalTransition(const Time& time);
virtual void externalTransition(const Time& time,
    const ExternalEventList& lst);
virtual void output(const Time& time,
    ExternalEventList& out) const;
virtual void confluentTransition(const Time& time,
    const ExternalEventList& lst);
virtual Value* observation(
    const ObservationEvent& event) const;
}
```

The listing 1 shows a simplified representation (without constructor, destructor, and other members) of the C++ class Dynamics. The Dynamics class permits to develop atomic model behavior. The five first functions represent the DSDE interface in VLE. The last function is a specific feature of VLE allowing observation of the state of an atomic model regardless of the simulation time. In VLE framework, each subclass of Dynamics must be compiled into a shared library. Developers of atomic DEVS models can share or reuse their models by exchanging C++ source code or binary code. However, this class and the formalism DEVS is not adapted for many modelers. Thus, we decide to add different modeling formalisms in VLE.

To develop models that change the structure of the model during the simulation, modelers develop C++ classes that inherits the class `Executive`. This class has the same interface as the previous `Dynamics` class. However, it offers some functions to manipulate the structure of the model (add and delete models, connections).

B. Modeling formalism

Modelers are not fluent with the DEVS formalism and they prefer using mathematical formalisms as ordinary differential equation or automaton. So, rather than force them to translate their models in the DEVS formalism, VLE proposes a set of **modeling formalism** to perform the development of model without knowledge of the DEVS formalism. These modeling formalisms are developed as subclasses of Dynamics.

Listing 2: E.g. API of the Euler extension.

```
Class Euler: public Dynamics {
public:
virtual double compute(const Time& time);
};
```

For example, the listing 2 shows the class `Euler`. This class proposes a numerical solver of ordinary differential equation (ODE). Users of `Euler` are not expected to manipulate directly DEVS formalism. Only one method is necessary to declare ODEs variables and to compute their gradients.

With VLE, modelers can develop their models with several modeling formalism. From ODE numerical solver, finite state automaton, cellular automaton etc. These modeling formalisms are developed using a DEVS BUS. In VLE framework, modeling formalisms are sub-classes of the Dynamics class (See listing 1. Modelers must inherit the modeling formalism class to develop atomic model. As for Dynamics, developers can share or reuse their multi-models by exchanging C++ source code or binary code.

C. Integrated Development Environment

VLE provided C++ classes to develop DEVS atomic models from modeling formalisms classes or from basic DEVS atomic model class. However, to model the DEVS models, modelers need to define a hierarchical networks of models. In order to simplify the use and the reuse of the C++ classes, the structure of the model is stored into a classical XML application. The XML application describes (i) the structure of the models, (ii) the parameters to initialize atomic models and (iii) the atomic models to observe. VLE proposes the software program GVLE to perform the edition of this XML application. GVLE is also an IDE to write C++ classes and to develop and compile shared library. For the users of the VLE environment, GVLE is certainly the most important application software.

D. Extra Software Programs

Statistical methods for parameter estimation, validation and sensitivity analysis are necessary for implementing and analyzing agro-ecosystem models [8]. VLE provides a package *rvle* that links VLE to the statistical language R [9] in order to perform models calibration and exploration.

Therefore, we proposed a web-oriented tool, designed to develop web applications around the models implemented on the environment. This tool is based on the development of a specific package: PyVLE that enables the use of VLE from Python applications. This package provides a set of functions that enable the management of VLE simulations from Python scripts.

We present, in this section, three options to improve collaborative aspects in the VLE environment. However these steps are not sufficient to address the problem of exchanging and reusing models. To address this problem, we propose a solution based on the packaging of models. Moreover, this solution greatly improves collaboration between users.

III. METHODS

VLE now offers a wide range of programs and libraries. Each of these components help to share expertise between researchers and other users. However, the exchange of models or data between modelers is still one major issue. If a modeler wants to reuse a model, it needs to copy, install, compile the source code of the model to reuse. New questions arise. How to take into account the evolution of the imported model ?

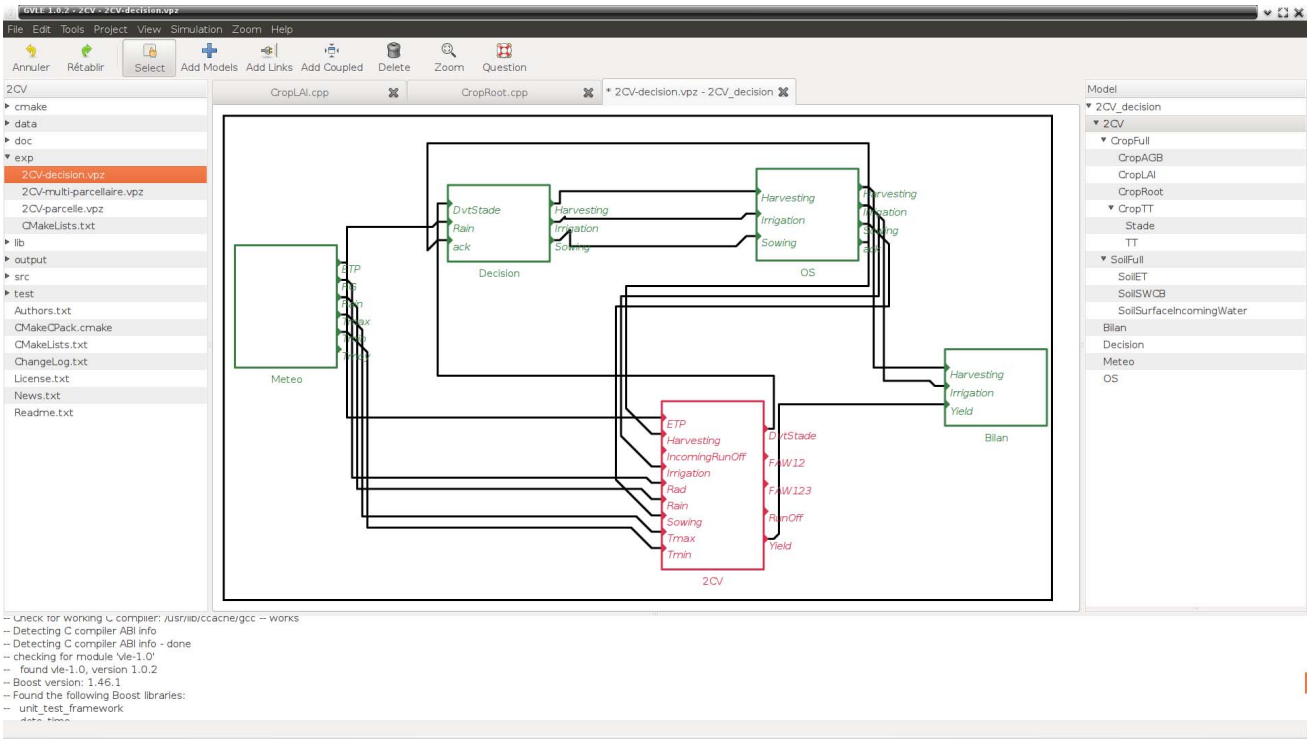

Fig. 1: The IDE of VLE: on the left of the picture, GVLE shows the current opened package. On the right, GVLE shows the content of the current experiment. In the middle, the current coupled model.

How to make compatibility remain at API, ABI level or at the DEVS level ?

The same problems have been resolved by the Free Open Source Software (FOSS) community and Linux distributions as Debian or RedHat. They develop package system manager such as `apt-get` or `yum`. In this paper, we propose to add a package system manager to VLE to improve the collaboration between users of VLE.

A. *Package system*

The package system must provide several features. Many of these characteristics are the same as in the Linux distribution such as Debian or Redhat [10]. The solution proposed in this paper is largely inspired by the works of the FOSS community, but adapted and restricted to the M&S in VLE environment.

Thus, a package is an archive of sources, data and documentation files with a description file. The description file describes the name, the version, the dependence to the version of VLE and three lists of packages that define the dependencies to build the package, the dependencies to run the package and the conflict dependencies. For example, the weather package in agronomic sytem, as no running dependence while the list to build the package contains the modeling formalism difference equation. We categorize the package:

- Model package provides models (atomic or coupled) and contains source code and binary code of the models, data and documentation.
- Formalism package contains a modeling formalism source, headers (to use as a subclass), binary code, data and documentation.
- Data package contains only data.
- Meta package contains only a description file with the list of dependency.

The format of the package is showed in listing 3. A package is a directory or an archive that allows the construction of model package, modeling formalism package, data package or meta package. The `data` directory stores input, `exp` stores the simulation experiments (the XML application), `src` stores the code sources of the models, `test` stores the unit test. VLE builds the `plugins` and `lib` directories to store respectively the compiled atomic models and the shared library from modeling formalism.

Listing 3: A part of the structure of a package in VLE

```
package/
 data/
 exp/
 src/
 test/
 lib/
 plugins/
 Authors.txt
 Description.txt
 License.txt
```

B. Description file

To define a package for VLE, users need to define a description file `Description.txt`. This description file exists in classical Linux distribution as Debian [11] or in R [9]. In this file, Two fields are necessary. A package name and a version. The package name must be unique in the distribution. The version number follows a strict representation. Indeed, in Debian [11] or RedHat package system, the version number is a string and it complicates the package systems. For the VLE community, we restrict the package version. The version must respects the regular expression:

Listing 4: Package version in regular expression.

```
[0-9]+\.[0-9]+\.[0-9]+\-[0-9].
```

For example, a package can have version a `1.0.0-0` or `1.2.3-7`. The third numbers define the version of the package. The last number defines the repository and integrator of the repository version. Integrator can correct package and release a package.

The VLE environment provides C++ development libraries. Thus, we define a policy to ensure stable API and ABI and allows developers to correctly reference VLE's API and ABI.

Listing 5: VLE version in regular expression.

```
[0-9]+\.[0-9]+\.[0-9]+\-[a-Z0-9]+
```

The first two numbers correspond to the major and the minor version. These numbers define the stable version of VLE. The patch version number, the third number, defines the number of release for the stable version of VLE. We ensure stable API and ABI for the all stable version of VLE. For example, the weather package compiled with VLE 1.0.0 needs to be updated and compiled to add compatibility with the VLE 1.1 while for VLE 1.0.7, the model is already compatible.

Listing 6: The description file of the model package is a text file

```
Source: weather-gen
Version: 1.0.0-0
Section: agronomic
Maintainer: quesnel@users.sourceforge.net
Homepage: http://www.vle-project.org
Build-Depends: vle (>= 1.0),
          differential-equation (>= 1.0)
Depends: weather (= 1.2.3), output-file (>= 1.0)
Conflicts:
Description: A weather generator.
```

The description file of the listing 6 shows the `weather-gen` (weather-generator) package. This package is built with `1.0`. It recommends the package *weather* in version `1.2.3` and the package *file* in version greater than `1.0`.

C. Distribution

Packaging the models is the first step of building a package system. The next steps are to develop tools and methods to build a distribution of all available packages. A **distribution** allows users to install and uninstall packages from a distant repository.

A distribution is a finite set of **packages rules**. Each package rule is a tuple of the form (p, B, D, C) where p is a package, B is a set a **dependency build clauses**, D is a set of **dependency clauses** for p. C is a set of **conflict clauses** for p. the dependency build clauses and dependency clauses must be present to compile and use the package p.

Each dependency clauses or dependency build clauses is a disjunction of packages $p_1|...|p_k$. A **dependency clause** requires that some packages from the set $\{p_1, ..., p_k\}$ be present. **Conflict clause** requires that package p' not present in order to install the package p. Thus, a valid installation is a subset of the packages in the distribution. For example, a set of packages from the distribution installed on a machine of the user community. To ensure functioning of the machine we require that **installation profile** of the machine be valid. A installation profile is valid if dependency clauses and conflict clauses are satisfied. A valid installation profile for a distribution in one that satisfies the dependency and conflict clauses of each package rule of the distribution.

- A dependency clause $p_1|...|p_k$ for package p is valid iff either p is not installed and some packages in the set $\{p_1, ..., p_k\}$ are present in the profile.
- A conflict clause is valid iff either p is not installed and p' is not installed.

The package system have two critical problem.

- To install the new package p given a distribution R and an installation profile P, the package system has to check if there is a profile P' containing p such that $P \cup P'$ is a valid installation profile for R.
- It may happen that a new package p cannot be installed because it is in conflict with some packages already installed on the machine. In this case, we must first uninstall some packages before attempting to install p. The task of the package system is then to identify a set of packages P' such that p can be installed on $P - P'$

To address these two problems, we can use a satisfiability problem (SAT) solver as proposed in the OPIUM package manager [12].

D. Distribution Workflow

The task of maintaining a package repository is difficult. It requires to take into account the evolution of all packages over the time, address the error reports from users, developers and the quality assurance (QA) teams. The figure 2 shows the workflow of the distribution. The distribution workflow needs the development of tools to automatize as much of this work as possible.

The role of the QA teams of the distribution is to coordinate bug filling, tracking the absence of maintainers, development of tools, running systematic checks on the entire archive and publishing. Thus, we provide several scripts to automatize the workflow.

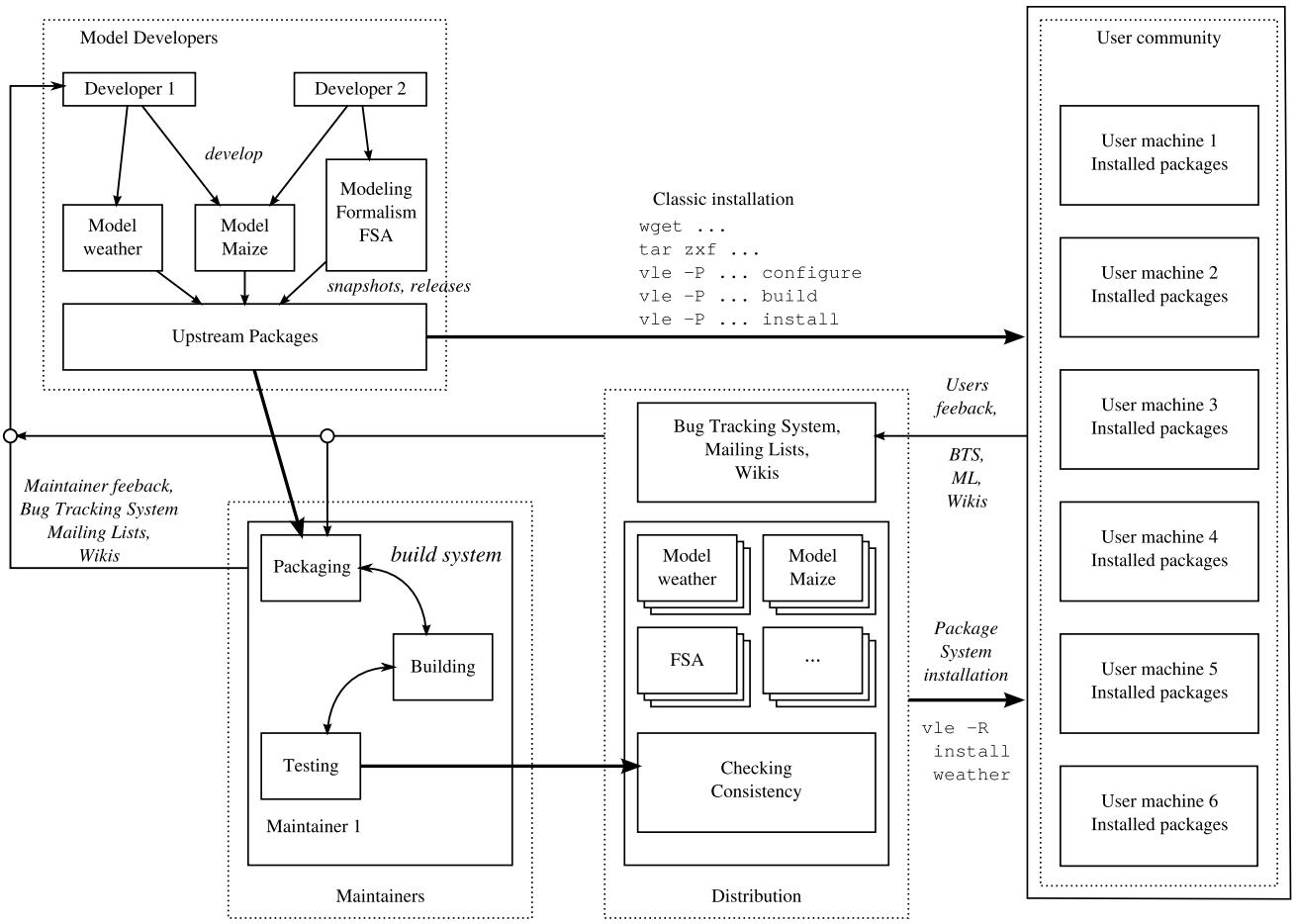

Fig. 2: Major flow of the VLE package system.

IV. Results

In March 2012, the package system manager is not completely available for an every day use. Several model packages begin to convert into the package format. The following modeling formalism have been converted into packages:

- Difference-Equation is a set of class to develop models with recurrence relation i.e. equation that recursively defines a sequence.
- Differential-Equation provides four classes to solve numerically ordinary difference equation (ODE): Euler, Runge Kutta, QSS1 [13] and QSS2 [14].
- FSA is a collection of classes to develop model based on finite state automaton: Moore, Mealy, FD-DEVS and Statechart [15].
- PetriNet provide and implementation of the High level Petri net.
- Agent permits to build tasks networks.
- CellDEVS is a class to develop cellular automaton [16].
- CellQSS is a subclass of the CellDEVS to solve numerically partial differential equation.
- Output is a set of plug-ins to write output of simulation in different file format.
- Examples is a set of examples that use the previous packages.

In this section we show the construction of a model based on several packages. The goal is to optimize the sowing date of maize according to a stochastic weather generator and an agent decision making.

The basic crop model uses the *FSA* and *Difference-Equation* modeling formalism. The first one is used to model maize stage (sowing, harvesting). The second one is used in several atomic models to model the processes of the maize model (Leaf Area Index (LAI), intercepted Photo-synthetically Active Radiation (PAR) and biomass). The *maize* model need to be coupled with a *weather* model that sends temperature and rain daily. The *maize-agent* package introduced a model that inherits of the *Agent* modeling formalism. This model permits to define tasks such as sowing, harvesting, irrigating, fertilization. In another package, we couple a pseudo random number generator (Mersene Twister [17]) *PRNG* and the weather model to build a *weather generator*. Finally, we define a package *maize-optim* that combine class from an optimization package and the *maize-agent* to build the complete model.

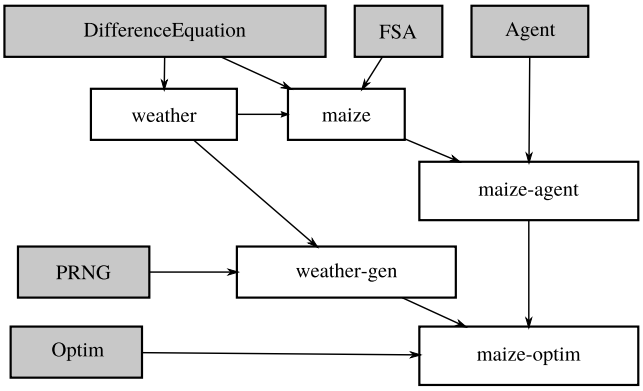

Fig. 3: An example of package dependencies with an agronomic model.

V. Discussion

For now, the package system is operational and packages can be downloaded and installed from a server. Only the dependence system is not implemented. The tools to perform integration of the distribution are also not yet available.

The use of package seems to simplify the construction of complex models by aggregating atomic models and data from other packages. However, we note that many modelers use modeling formalism package and generic model packages such as the *weather* or the *glue* packages. Users do not prefer to create dependency with final package.

We think it is not related to the package systems, but to VLE functioning and to the design model. Indeed, developing a generic dynamic model is complicated. In DEVS and VLE, a model can be seen as a black box in Systems Approach or systemic world view. Events, from another models, can perturb the model any time. These perturbations can arrive in an unexpected state or transport malformed data. Considering these events, developing a robust model becomes a long task. Modelers prefer to use generic models with robust designs.

VI. Conclusion

In this paper we define a package system management for the VLE modeling and simulation environment. The package system enables to share data, models source code and shared libraries. It allows the use and the reuse of models developed by another users in VLE community. It improves the global quality of the model and simplify the maintenance of the models. However, the package system needs to have computer engineers (i) to perform *release* of the distribution and (ii) to integrate and maintain package of the community. For now, the package system begins to work perfectly but there are still missing features such as the checking of packages dependencies.

Unfortunately, the package system proposed in this paper depends on the VLE software and its specification. However, we adapt the package systems to VLE requirements. For example, an atomic model must be compiled into shared library or into a plug-in before running a simulation. Thus, we need to define a package system that can compile C++ source code.

In this paper, we focus the package system to the development of the behaviour of atomic modes. However, it may also be necessary to help modelers to build hierarchy of models from the package system. For now, we provide only a way to export sub-classes of the `Executive` class (which allows drastic changes in the structure of the model). In future development, we want to develop the possibility to import and export some part of structures of models from another package.

References

[1] G. Quesnel, R. Duboz, and E. Ramat, "The Virtual Laboratory Environment – An operational framework for multi-modelling, simulation and analysis of complex dynamical systems," *Simulation Modelling Practice and Theory*, vol. 17, pp. 641–653, April 2009.

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

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

[4] F. J. Barros, "Dynamic Structure Multiparadigm Modeling and Simulation," *ACM Transactions on Modeling and Computer Simulation*, vol. 13, no. 3, pp. 259–275, 2003.

[5] G. A. Wainer, "Cd++: a toolkit to define discrete-event models," *Software, Practice and Experience. Wiley*, vol. 32, no. 3, pp. 1261–1306, November 2002.

[6] J. J. Nutaro, *Building Software for Simulation: Theory and Algorithms, with Applications in C++*. Wiley Publishing, 2010.

[7] F. Bergero and E. Kofman, "PowerDEVS: a tool for hybrid system modeling and real-time simulation," *SIMULATION*, Apr. 2010.

[8] D. Wallach, D. Makowski, and J. Jones, *Working with dynamic crop models: evaluation, analysis, parameterization, and applications*. Elsevier, 2006. [Online]. Available: http://books.google.fr/books?id=nG7DEXen9QAC

[9] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2011, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org/

[10] J. Boender, R. Di Cosmo, J. Vouillon, B. Durak, and F. Mancinelli, "Improving the quality of gnu/linux distributions," in *Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*, ser. COMPSAC '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 1240–1246.

[11] M. F. Krafft, *The Debian System: Concepts and Techniques*. San Francisco, CA, USA: No Starch Press, 2005.

[12] C. Tucker, D. Shuffelton, R. Jhala, and S. Lerner, "Opium: Optimal package install/uninstall manager," in *Proceedings of the 29th international conference on Software Engineering*, ser. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 178–188.

[13] E. Kofman and S. Junco, "Quantized State Systems. a DEVS Approach for Continuous Systems Simulation," in *Transactions of SCS.*, vol. 18, 2001, pp. 123–132.

[14] E. Kofman, "A second order approximation for devs simulation of continuous systems," *Journal of the Society for Computer Simulation International*, vol. 78, no. 2, 2002.

[15] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Programming*, vol. 8, pp. 231–274, 1987.

[16] G. A. Wainer and N. Giambiasi, "Application of the Cell-DEVS paradigm for cell spaces modelling and simulation," in *Simulation*, vol. 76, 2001, pp. 22–39.

[17] T. Matsumoto and T. Nishimura, "Mersene twister: A 623-dimensionally equidistributed uniform pseudorandom number generator," in *ACM Trans. on Modeling and Computer Simulation*, vol. 8, January 1998, pp. 3–30.