A Proposed Evolution of DEVSimPy Environment Towards Activity Tracking

Jean François Santucci, Laurent Capocchi

A Proposed Evolution of DEVSimPy Environment Towards Activity Tracking

ACTIMS Workshop, May 28 - June 1 2012, Cargese, Corsica (France)

J.F. Santucci, L. Capocchi

This paper deals with the potential evolutions of the DEVSimPy environment involving Activity Tracking (AT) concepts. The DEVSimPy environment is being developed at University of Corsica in order to facilitate the modeling and the simulation of dynamic systems described with the DEVS formalism. Currently, DEVSimPy can be used as an AT software providing some features like simulation profiling or step-by-step simulation. However, DEVSimPy has been developed in an object-oriented way using as far as possible Design Patterns. This property makes easy the introduction of new concepts or functionalities in order to extend the DEVSimPy towards a real AT software for DEVS systems.

1 Introduction

This presentation concerns the potential evolutions of the DEVSimPy framework involving Activity Tracking (AT) concepts. This framework is being developed at University of Corsica using the Python language. DEVSimPy is an open source project under GPL V3 license and the SPE research laboratory team supports its development. It uses the wxPython User Interface library, the Python Language and the PythonDEVS API [1]. The DEVSimPy environment is being developed in order to facilitate the modeling and the simulation of dynamic systems described with the DEVS formalism. Currently DEVSimPy can be used as an AT software providing some features as simulation profiling or step-by-step simulations. Furthermore DEVSimPy has been developed in an object oriented way using as much as possible Design Patterns. This feature allows an easy introduction of new concepts or functionalities in order to extend the DEVSimPy framework towards a real AT software for DEVS systems. The goal of this paper is therefore threefold:

- Over-viewing of the DEVSimPy framework: An overview of the object oriented architecture will be presented and the following features will be

pointed out: (i) how the re-usability of models is achieved; (ii) how different types of simulation is allowed and (iii) how it is easy to implement plug-ins in order to extend the semantic of DEVSimPy.

- Highlighting AT in actual DEVSimPy framework: Some AT features already implemented in the DEVSimPy framework. will be described. In particular the implementation of a special plug-in named 'Blink' will be pointed out. The aim of this plug-in is to detect which atomic model is currently active and indicate interactively with the user all the information concerning the simulation process. A 'No Time Limit' option has been defined and implemented based on the concept of starting and finishing activities. The main idea was to implement two methods allowing to: (i) initiate a thread corresponding with the beginning of a simulation process; (ii) detect the fact that no activity is currently present in a given simulation process. A profiling option allows the user to obtain a set of statistical information about the activity of models which have been collected during the simulation.
- Envisioning evolution of DEVSimPy framework in order to deal with AT concepts: the potential extensions of DEVSimPy will be explored in order to implement the concepts stemming from AT domain [6]. The oriented object implementation of DEVSimPy based on the use of design patterns facilitate the introduction of new concepts in order to extend the DEVSimPy framework. Addition of extensions of DEVSimPy could be easily performed own to : (i) a simulator based on AT using the Design Pattern Strategy; (ii) information about activity tracking during the simulation using a dedicated plug-in.

The rest of this paper is organized as follows : the next part introduces the DEVSimPy framework. The object oriented architecture is presented and the main functionalities are described. Section 3 deals with AT notions already introduced in the DEVSimPy framework. The main features already implemented will be detailed : the 'Blink' plug-in, the 'No Time Limit' option, the start and finish methods and the profiling option. In section 4 the basic ideas allowing to extend the DEVSimPy framework using AT features are given.

2 DEVSimPy Environment

DEVSimPy is an open source project initiated by the Modeling and Simulation team of the SPE (Sciences Pour lEnvironnement) laboratory (University of Corsica). The aim of this software is to provide developers a collaborative modeling and simulation (M&S) framework in Python language [2]. It uses the wxPython library which is a blending of the wxWidgets C++ class library with Python. The DEVSimPy M&S kernel is based on the PythonDEVS API which offers a consistent and coherent set of classes in order to

construct a modular system and to achieve its hierarchical simulation. Initially, the idea of the DEVSimPy project was to decorate the PythonDEVS kernel with a graphical user interface (GUI) specially to be able to handling models in a visually dynamic way. All DEVS models implemented with the PythonDEVS API are admissible into a DEVSimPy diagram as a graphical box with ports. Basically, a DEVSimPy library is composed with PythonDEVS models which can be instantiated using a drag and drop operation. Subsequently, many customizable aspects have been added to the user interface in particular to simplify the coupling between models, to save them in a specific format or to edit the corresponding code. If the PythonDEVS file can be considered as the default model handled by DEVSimPy, .amd and .cmd files have been introduced to represent respectively atomic and coupled DEVS models. More specifically, these are compressed files containing separately a behavioral and graphic file. The behavioral file implements DEVS specification with the PythonDEVS rules and the graphic file allows the view of the model in DEVSimPy interface. The approach consisting to separate the behavior and the view of model provides a way to handle these two aspects in a efficient way. In fact, when it is necessary to change or to permute a behavior of model, the view may stay the same. Finally, the high-level coupled model (called diagram) which includes models (.amd, .cmd, .py) has a specific format named dsp. The developers have the possibility to share dsp associated with libraries in order to simulate diagrams which have been already built.

The philosophy of DEVSimPy is to be an open, extensible and participative environment for the developers. A plug-in manager is proposed in order to expand the functionalities of DEVSimPy allowing their enabling/disabling through a dialog window. DEVSimPy provides a set of existing plug-ins allowing the verbose simulation, the double click event overwriting with the specification of a new action, the hierarchical view of the simulation objects using a tree graph, the representation of GIS in Google Earth [5] and even more. The list of active plug-ins is saved in a configuration file loaded during the starting of DEVSimPy.

2.1 Architecture

From the perspective of a software engineering, DEVSimPy is implemented in a object oriented way with the use of design patterns to insure a perfect software evolution. Furthermore, the Model-View-Controller (MVC) has been used to organize the interaction with the users and to split the software architecture into two entities: the PythonDEVS kernel and the DEVSimPy engine. This separation help to facilitate the independent development, testing, and maintenance of DEVSimPy. Moreover, the "Strategy" pattern is used to allow the definition of interchangeable simulation algorithm family. Users can choose at run-time between the classic hierarchical DEVS simu-

lator, the flat version, or other simulators proposed by the developers. The use of the Strategy pattern insures the indecency of simulator algorithms and provide a way to separate algorithms into classes that can be plugged in at runtime. According to the fact that Python language allows the introspection of its objects, DEVSimPy provides the possibility to change the behavior of models during the simulation process. This feature has been used for the fault simulation of system by injection. In [3,4], DEVSimPy is used to facilitate the fault injection into a neural network based discrete event model of electrical machines for its diagnosis.

2.2 Functionalities

A modeling and simulation software should be able to propose a set of functionalities dedicated to facilitate handling of models and to automate as far as possible the simulation process. DEVSimPy has been build in order to manipulate PythonDEVS models in a graphical way using drag and drop function to instantiate DEVS models and save them in a specific file with a .dsp extension. DEVSimPy offers basic functionality like the copy/cut/past of models, the edition of model properties (graphical and behavioral attributes), the coupling between models (with drag and drop), the export of models in a library (with .amd or .cmd format resp. for atomic and coupled model), the automatic simulation by using a simple click on a button. In addition, DEVSimPy has been build with a set of extra functionalities making it a powerful software to model and simulate DEVS models:

- With DEVSimPy, the modeler can use plug-ins in order to control and extend the behavior of a set of models (called general plug-in)or a simple model (called specific plug-in). The user controls all of these plug-ins by using a plug-in manager in charge of enable or disable them. For example, DEVSimPy proposes a general plug-in which is in charge to blink the models whose the transition functions are executed. There is also a specific plug-in in charge of the double right click overwriting by disabling the default functionality which displays the property manager.
- An another functionality which makes DEVSimPy unique is the possibility of changing the code of model during simulation. The modeler can decide to suspend simulation to change the code of model and restart the simulation to include the new behavior of the model. This functionality is often used when visualizing the simulation results and when the modeler decides to interact with the model and wants to change the behavior of one or more models to control the simulation results.
- With DEVSimPy, the modeler can export or import models from dynamic libraries composed by local or web models. When models are stored in a local directory, the importation is made knowing the absolute path of .amd, .cmd or .py files. When models are stored on the

Web server (http(s)or ftp), the importation is made from the url of the model file. This functionality is useful when developers want to manage DEVSimPy files in a collaborative way. Moreover the distribution of Web libraries is much easier than the local libraries. In addition, all of these functionalities make DEVSimPy a powerful software for the modeling and simulation of hybrid system due to the possibility of using dynamic libraries composed with connectible and compatible models.

- An interesting functionality in DEVSimPy is the profiling option once the simulation is over. The developers can analyse the execution time, the number of transition function execution for all DEVS atomic models. This last functionality is often use to measure a certain notion of model activities. This functionality can be also used during the simulation.

The set of these DEVSimPy functionalities are possible with a software architecture based on a modular class diagram using a MVC design.

3 Activity Tracking according to DEVSimPy

3.1 Activity Tracking Concept

Basically, the activity notion for a DEVS system can be viewed in two ways: in the modeling part or in the simulation part (figure 1). In the modeling part, a DEVS model activity can be considered as the number of transition functions executions. In the simulation part, an activity measure can be a quantification of an entity (like the number of CPU cycles) used to execute the model.

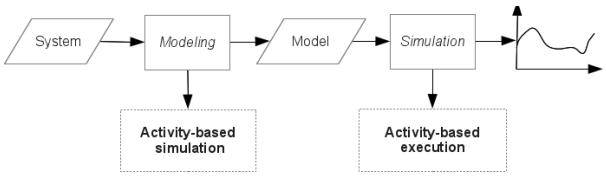


Figure 1: Notion of activity in the modeling and simulation part.

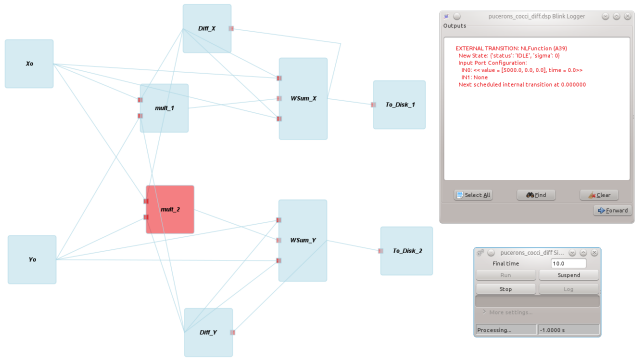
When the simulation activity is tracked in the simulation part, a main goal could be to analyze the computation simulation process (considering a measure like CPU time consumption) to improve it by changing or adapting its algorithm. For the modeling part, a goal could be to calculate the number of model execution in order to improve its modeling.

3.2 DEVSimPy functionalities involving Activity Tracking

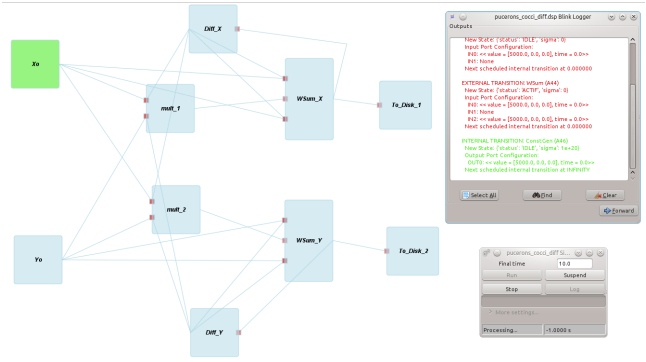
According to the DEVS formalism, the state of a model changes when an internal or an external event occurs during the simulation. The simulation

process unfolds as long as at least one atomic model has its internal transition function which must be active. The simulation is over when the time advance of models is infinity. This condition is not unique and can be coupled with another constraint specified by the modeler. A simulation time limit can be imposed by the modeler in order to end the simulation process. DEVSimPy offers a 'No Time Limit' option to disable this one. If this option is true, the simulation process is active if at least one model is active. In addition, DEVSimPy allows the implementation of two functions (advisable but not mandatory): the "start" function and the "finish" function. These functions can be used to specify the behavior of atomic models respectively at the beginning and the end of the simulation. The activation of these functions is automatic in the simulation part in so far as they are implemented by the modeler.

DEVSimPy includes a plug-in manager to extend its functionalities called 'Blink'. The plug-in 'Blink' is proposed to visualize the activity of models during the simulation (Figure 2). It is based on a step by step approach and filled each active model with a color which depends on the executed transition function.



(a) Blink starting the simulation.



(b) Blink during the simulation.

Figure 2: Example of activity visualization with blink plug-in.

For instance,the model becomes red when intercepting a external event and then executes its external transition function (see figure 2(a)). It becomes green when a model performs its internal transition function (see figure 2(b)). With this plug-in, the modeler can also find (in a associated dialog window shown in both figure 2(a) and 2(b)) details about messages composition, the evolution of models state or the messages intercepted/sent on their input/output port during the simulation.

4 Proposed Evolution of DEVSimPy

We describe in this part the basic idea we envision to extend the DEVSimPy framework by implementing AT concepts. The idea is to propose an enhanced simulation based on AT concepts. In order to improve the performance of the simulation of complex DEVS systems in terms of CPU time we propose to develop an approach based on AT. In order to clarify the activity notion we have to point out that two level of activity can be highlighted as introduced in part 3.1: (1) activity at the model level; (2) activity at the computing level. The first one refers to the notion of activity linked to: (i) the number of internal discrete-events (internal function activation by atomic models) over a simulation-time period; (ii) the number of external discrete-events (messages exchanged by atomic models) over a simulation-time period. The second one concerns the CPU time consumption linked with the processing of atomic models involved in the studied models. The idea is to measure the CPU time associated with the different atomic models and to adapt the simulation according to the result of the measurement of the computational activity. Two steps are necessary to implement this improvement: (1) measurement of the computational activity associated with atomic models; (2) re-organization of the simulation architecture according to the previous obtained measurement. The re-organization will consist in selecting the best simulation algorithm in order to improve the CPU time when performing the simulation. The implementation of simulation algorithms in DEVSimPy has been done using the Design Pattern called Strategy [7]. The Strategy Design Pattern allows to: (i) define a family of algorithms, encapsulate each one, and make them interchangeable; (ii) capture the abstraction in an interface, bury implementation details in derived classes. The Strategy Design Pattern we have defined allows the user to choose between different simulation algorithms. We already have implemented three different kinds of simulation: (i) classical hierarchical DEVS simulation [8]; (ii) direct coupling simulation; (iii) Concurrent DEVS simulation. The figure 3 illustrates the actual implementation of the Strategy Design Pattern allowing the DEVSimPy framework to run the hierarchical simulation (Hierarchical), direct coupling simulation (DirectCoupling) and concurrent simulation (Concurrent). We plan to add a fourth strategy corresponding to AT simulation (it is called

ActivityTracking).

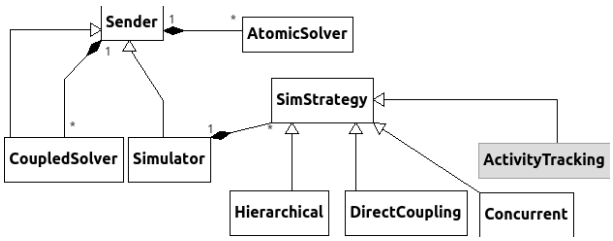


Figure 3: UML diagram pointing out the implementation of Design Pattern strategy.

The measurement of CPU time consumption will be performed using the profiling option presented in part 3. The integration of the measurement of activity within the simulation architecture is performed according to figure 4. The measurement of activity is performed using the profiling option already implemented in the DEVSimPy framework. This measurement is then used in order to dynamically choose between the best simulation algorithm as it is shown on figure 4. The implementation of different simulation algorithms organized according to the Strategy Design Pattern will allow the selection of the best algorithm while the simulation is conducted.

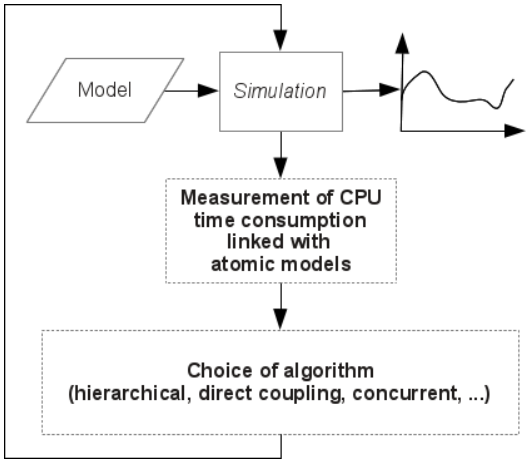


Figure 4: Integration of activity measurement with simulation algorithm.

The dynamic switching between the different simulation algorithms in order to improve the CPU time consumption according to the activity measurement is being implemented own to both the Strategy Design Pattern concept and the intrinsic dynamic programming feature of the Python language.

5 Conclusion and Perspectives

This paper deals with the links between AT concepts and DEVSimPy framework. After a brief description of the DEVSimPy environment we first pointed out how AT concepts are already involved in the DEVSimPy. Two main examples have been given : the 'No Time Limit' option and the 'Blink' plug-in. Then we present the main ideas we envision in order to propose extension of the DEVSimPy framework based on AT concepts : (i) implementation of a simulation algorithm based on AT ; (ii) definition of a plug-in allowing AT measures to be obtained during simulation of DEVS models. Own to the DEVSimPy architecture the extensions can be easily implemented and furthermore using the numerous already existing libraries the developers will able to exploit the new capabilities stemming from AT. Another original perspective of DEVSimPy to activity is to relate "Activity-based execution" to "Activity-based simulation" by differencing the number of transitions from the corresponding execution times of the transitions.

References

[1] J. S. Bolduc and H. Vangheluwe, The modelling and simulation package PythonDEVS for classical hierarchical devs., MSDL Technical Report MSDL-TR-2001-01. Montreal, Quebec, Canada: McGill University, (2001)

[2] L. Capocchi, J. F. Santucci, B. Poggi, C. Nicolai, and others, DEVSimPy: A Collaborative Python Software for Modeling and Simulation of DEVS Systems, in 2011 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 170175, (2011)

[3] L. Capocchi, S. Toma, G. A. Capolino, F. Fnaiech, A. Yazidi, Wound-Rotor Induction Generator Short-Circuit Fault Classification Using a New Neural Network Based on Digital Data, in Proceedings of the 8th IEEE International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives will be held during, September 5-8, 2011, Bologna (Italy), ISBN 978-1-4244-9302-9, IEEE Catalog Number CFP11SDE-USB

[4] S. Toma, L. Capocchi, D. Federici, A New DEVS-Based Generic Artificial Neural Network Modeling Approach, in Proceedings of The 23rd European Modeling and Simulation Symposium (Simulation in Industry), Rome, Italy, September 12-14, 2011.

[5] J. F. Santucci, L. Capocchi, Visualization of Folktales on a map by coupling dynamic DEVS simulation within Google Earth. SIMULTECH 2011, Noordwijkerhout : Netherlands.

[6] A. Muzy, F. Varenne, B. P. Zeigler, J. Caux, P. Coquillard, L. Touraille, D. Prunetti, P. Caillou, O. Michel, and D. R. . Hill, Refounding of Activity Concept? Towards a Federative Paradigm for Modeling and Simulation.

[7] E. Gamma,R. Helm,R. Johnson, J.Vlissides, Design Patterns Elements of Reusable Object Oriented Sofware, Addison-Wesley, 1995.

[8] B.P. Zeigler, H. Praehofer , T.G. Kim, *Theory of Modeling and Simulation, 2nd Edition* (Academic Press, jan 2000) 510 pages.