MULTILAYERED DEVS MODELING AND SIMULATION IMPLEMENTATION VALIDATION ON A CONCRETE EXAMPLE: PREDICTION OF THE BEHAVIOR OF A CATCHMENT BASIN

Emilie Broutin
Paul Bisgambiglia
Jean-François Santucci

University of Corsica, SPE
CNRS, UMR N°6134
Campus Grimaldi
20250 CORTE
Email : {broutin | bisgambi | santucci}@univ-corse.fr

KEYWORDS
DEVS, Multilayer, Python-DEVS, reusability, catchment basin, level of abstraction

ABSTRACT

Modeling complex natural system is a difficult task requiring the cooperation between several specialists. Owing to these specialists we can obtain separate precise models but it is often necessary to interconnect them in order to solve a given problem. However this interconnection raises two kinds of problems: (a) data may be expressed into different units according to the models; (b) time units may be different when running the different models. We solve these problems by using a special component.

1. INTRODUCTION

Modeling a complex system is a collaborative work between specialists from various expertise areas which results in the integration of a set of detailed models which have to deal with a great number of data. No problems occur until we need to connect these models to each other. But serious problems may appear during data exchange between models because data from a model may not fit to another model; temporal problems may also arise because of models working with the different time units. We have proposed a framework allowing to efficiently connect models in [1]: using the DEVS (Discrete Event Specification) formalism we have been able to define a new component called Assembly Model component. It contains conversion and temporal functions allowing to solve the two previously introduced problems. This paper will deal with a PythonDEVS [3] software implementation of the concepts of multilayered modeling and simulation already presented [1] and with the validation of the implemented software using a concrete application concerning the prediction of the behavior of a catchment basin. Such an application requires the interconnection of models developed separately and will be a nice validation example in order to describe and validate the concepts associated with the newly introduced Assembly Model component. The outline of the paper is the following. After an introduction which will set up the problem, the second section will briefly summarize the DEVS formalism, section 3 will presents the multilayer concept already presented in [1] which allow to interconnect different models using the Assembly Model component. Section 4 will detail the main classes involved in order to implement the previously summarized concepts before a brief presentation of the PythonDEVS implementation software of the multi-layered modeling and simulation. In section 5 we will present in detail how the Assembly model component involved in multilayered simulation can be used in order to deal with the prediction of the behavior of a catchment basin. We will first briefly describe two models developed by specialists required in order to perform precise simulations. The first one [2] defined by specialists belonging to the hydrological domain allows to model the behavior of a catchment basin according to rainfall. However in order to take into account the presence of snow a second model [2] defined by climatologists is also needed. We will then describe the software implementation of the two models, their interconnection using an Assembly Model component and the obtained results. Finally concluding remarks will permit to summarize the presented work as well as the future work we envision.

2. DEVS FORMALISM

DEVS formalism was created by Professor Zeigler [1] [2] [3] [4] [5] allowing to model a discrete event system. Two kinds of models are defined: 1) basic models from which larger ones are built, and 2) coupled models which describe how these models are connected together in hierarchical fashion. Basic models (called atomic models) are defined by the following structure:
CA = < X, S, Y, δint, δext, λ, ta> where:

(i) X is the set of input values;
(ii) S is the set of sequential states;
(iii) Y is the set of output values;
(iv) δint is the internal transition function dictating state transitions due to internal events ;
(v) δext is the external transition function dictating state transitions due to external input events ;
(vi) λ is the output function generating external events at the output, and
(vii) ta is the time-advance function which allows to associate a life time to a given state.

Proceedings 24th European Conference on Modelling and Simulation ©ECMS Andrzej Bargiela, Sayed Azam Ali David Crowley, Eugène J.H. Kerckhoffs (Editors)
ISBN: 978-0-9564944-0-5 / ISBN: 978-0-9564944-1-2 (CD)

The behaviour of an atomic model is illustrated as follows: the external transition function describes how the system changes state in response to an input. When an input is applied to the system, it is said that an external event has occurred. The next state s' is then calculated according to the current state s. The internal transition function describes the autonomous (or internal) behaviour of the system. When the system changes state autonomously, an internal event is said to have occurred. The next state s' is therefore calculated only according to the current state s. The output function generates the outputs of the system when an internal transition occurs. The time advance function determines the amount of time that must elapse before the next internal event occurs, assuming that no input arrives in the interim.
An atomic model enables us to specify the behaviour of a basic element of a given system.
A coupled model indicates how to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component of a larger coupled model, thus giving rise to hierarchical construction. A simulator is associated with the DEVS formalism in order to execute a coupled model's instructions so as to actually generate its behaviour. The architecture of a DEVS simulation system is derived from the abstract simulator concepts (Zeigler and al. 2000) associated with the hierarchical and modular DEVS formalism. The abstract simulator allows the definition of a simulation tree whose root element is dedicated to the time advance management.

3. MULTILAYERED DEVS ARCHITECTURE

One of the most difficult tasks in the field of modeling and simulation of complex systems is to choose a good level of detail. In all domains, models are built at a precise abstraction level. The abstraction level of a model determines the amount of information that is contained in the model (figure 1). As presented in figure 1 the quantity of information in a model decreases with the abstraction levels: a model described at a low abstraction level will contain more information than a model described at a higher abstraction level [16].

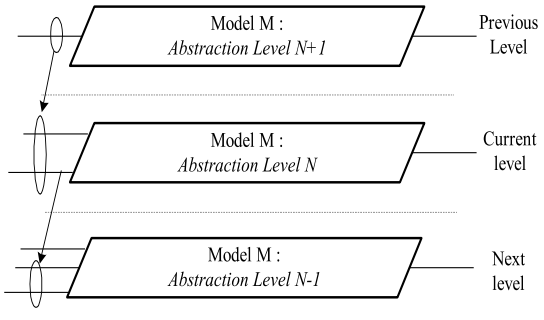

Figure 1: Abstraction Hierarchy

Well defining the abstraction level is an important step in modeling. A model described according to several abstraction levels is called a "hierarchical model". Let us call abstraction hierarchy such a notion of hierarchy.
We have to point out that this concept of abstraction hierarchy is quite different to the hierarchy of description inherent to the DEVS formalism. The notion of coupled models involving atomic or coupled models allows the definition of models using a hierarchy of description. Such a notion of hierarchy is only a mean to easily define models. In that case a flattened model can be easily generated from this hierarchy of description. However it is quite impossible to easily derive a flattered model when dealing with models involved in an abstraction hierarchy since data transfers have to be performed.
The presented work will allow to deal with models which have been designed according to different levels of abstraction. We have been able to define a multilayered DEVS architecture in order to perform the communication between models defined at different abstraction levels.

First of all we have studied HLA [15] and GDEVS [6]. These distributed architectures helped us to set up the time management of the simulation due to the similarity of the problems (especially in the next version of the multilayered architecture that will propose a new definition of the simulator including the modifying time management algorithm). However we cannot use HLA or GDEVS for our problem because one of our objectives is reusability of models. With the two-mentioned architectures we must modify and re-write the models because of the data unit problem.

The multilayered DEVS architecture [1] is based on the DEVS formalism [8]. This is an extension of the classical DEVS formalism that leans on the definition of a coupled model called Assembly Model involving two kinds of atomic models. The goal of our work is to allow an easy interconnection of DEVS models which have been defined separately and which were not dedicated to communicate with each other.

Because we have been able to solve the interconnections of models defined by specialists coming from different domains using classical DEVS atomic and coupled we did not need to modify the DEVS abstract simulator [9,10,11].

We differentiate two kinds of models:

- Behavioural models, which describe the dynamic of the systems required in order to solve a given problem i.e. the models defined by different specialists.
- Assembly model: it is the central point of the proposed formalism. Each behavioural model is linked to the assembly model and all the shared data are processed by the Assembly Model in order to performed data exchange between behavioural models.

During the data exchange three kinds of problems may occur:

- Temporal type of problems: the models involved in the study of a given phenomenon may not require data at the same time unit. For example a model may deal with seconds as time units while another one may require hours. This involve that a conversion will be necessary in order to keep the consistency of the data during the simulation

- Spatial type of problems: data issued from a geographic area associated with a model involved in the study of a given phenomenon may correspond to a larger or a smaller one in another model depending on the scales that have been used in both cases.
- And finally abstraction type of problems: Data involved in the different models required in order to study a given phenomenon may be at different levels of details. For example a model may deal with variables expressed with meters as units while another one may require variables expresses in kilometers.

In order to solve these problems, the Assembly model provides some conversions functions. In [1] we have presented a detailed version of the architecture with a formal representation.
Therefore we have slightly modified our original proposition. The most important element in the assembly model is the model called DRIV described more precisely in [1]. This component has three features:

- Time management: this feature is currently managed by the Root component; we plan to delegate this feature to the DRIV component in order to resolve the temporal problem. Because of the complexity of this feature, it will be detailed later in another paper.
- Management of the Input/Output : The DRIV component is in charge of the read and write order on the STO model. The conversion functions are called in the DRIV model instead of in the STO model as it is planed at first [1]. The DRIV component also centralizes the declarations of the data type and unit made by the authors of the different behavioural models. These declarations are required for the definition of the conversion functions.
- Management of the conflicts: The DRIV component contains a priority list for the resolution of the conflicts that may occurs during the data exchange.

The next section is devoted to the software implementation of the Assembly model while in section 4 we will detail the software implementation of a concrete example.

4. PYTHON-DEVS AND DEVSIMPY IMPLEMENTATION OF THE ASSEMBLY MODEL

In sub-section 4.1 we first briefly present the Python-DEVS software and the DEVSIMPY framework which has been used in order to implement the Assembly Model concepts. The sub-section 4.2 describes the implementation of the Assembly Model.

4.1. Python-Devs and DEVSIMPY

Python-Devs is a software implementation [3,12] of the DEVS formalism made by Jean-Sébastien Bolduc and Hans Vangheluwe using the python language. It provides two files:

- DEVS.py that contains the definition of the DEVS models i.e. the atomic and the coupled model.
- Simulator.py that implements the simulator engine.

Four classes are described:

- Port Class contains the common elements used by a DEVS port.
- BaseDEVS Class contains the elements use by both the coupled and the atomic model: the input and output port for example.
- AtomicDEVS and CoupledDEVS classes respectively describe the atomic and coupled DEVS model elements. Methods are added in these two classes. For instance the peek and poke methods of the AtomicDEVS class are used for sending and receiving a message on a port. In CoupledDEVS model, methods were also added for adding models or connecting ports. Furthermore a select method was added in order to define the priority between models if two events occurred at the same date.

A complete explanation of the Python-Devs

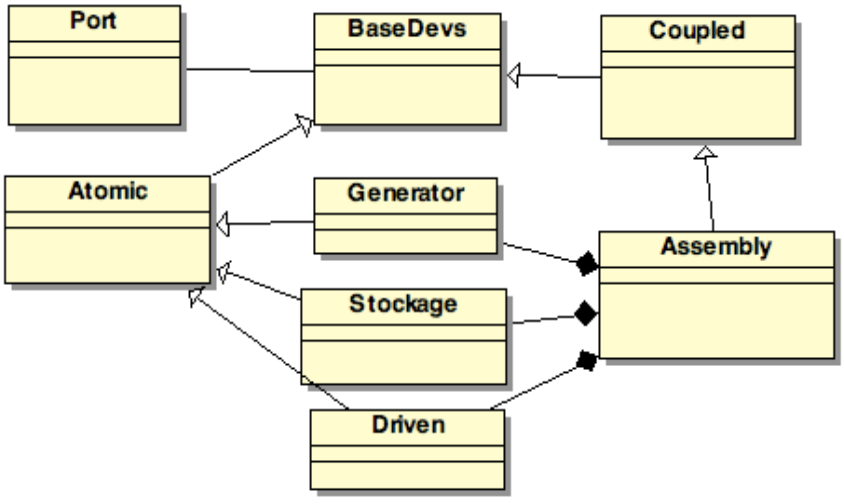

Figure 2 Class diagramm of the multilayered architecture

implementation can be found in [12] and the package can be downloaded in [3].
From Python-Devs Laurent Capocchi created a graphical environment. This environment provides a simple way for creating DEVS models : with a simple drag and drop an atomic or a coupled model can be created and clicking on it allows a window to appear for writing or modifying methods. The simulation is performed by running the Python-DEVS simulator.
A complete description of this graphical interface will be published soon.

4.2. Implementation of the ASSEMBLY MODEL using PYTHON-DEVS and DEVSIMPY

We added four classes to the original Bolduc and Vangheluwe definition of Python-Devs. Figure 2 presents the new class diagram.
We added a Generator class; its role is the creation of events. This class inherits from the Atomic Class.
We also implemented a DRIVEN class that inherits from atomic Class; this class represents the Driven model and its role is the reception and the transmission of data from/to the stockage models or the behavioural models.
A Stockage class has been also added; its role is the stockage of data. It inherits from the Atomic Class.
Finally we added a class called Assembly; this class inherits from the CoupledDEVS Class. This coupled model will allow to implement the interconnection of instances of DRIVEN, STOCKAGE and Generator classes.
Figure 2 present the class diagram of the multilayered architecture.

5. VALIDATION THROUGH A CONCRETE EXAMPLE

We choose a concrete example for the validation: the prediction of the hydrologic behavior of a catchment basin.
We will present in sub-section 5.1 the models involved in order to describe the hydrologic behavior of a catchment basin. Sub-section 5.2.

5.1. Hydrologic behavior of a catchment basin.

The hydrologic behavior of a catchment basin can be obtained by interconnecting two models: a snow model and a watershed model.
Among all of the existing watershed modeling we have select the GR3J model which has the advantages to be quite simple but also quite precise. GR3J is an hydrological model for the study of catchment. It performs good results by using a representation of the rainfall-runoff process as simple as possible and depending on very few parameters.

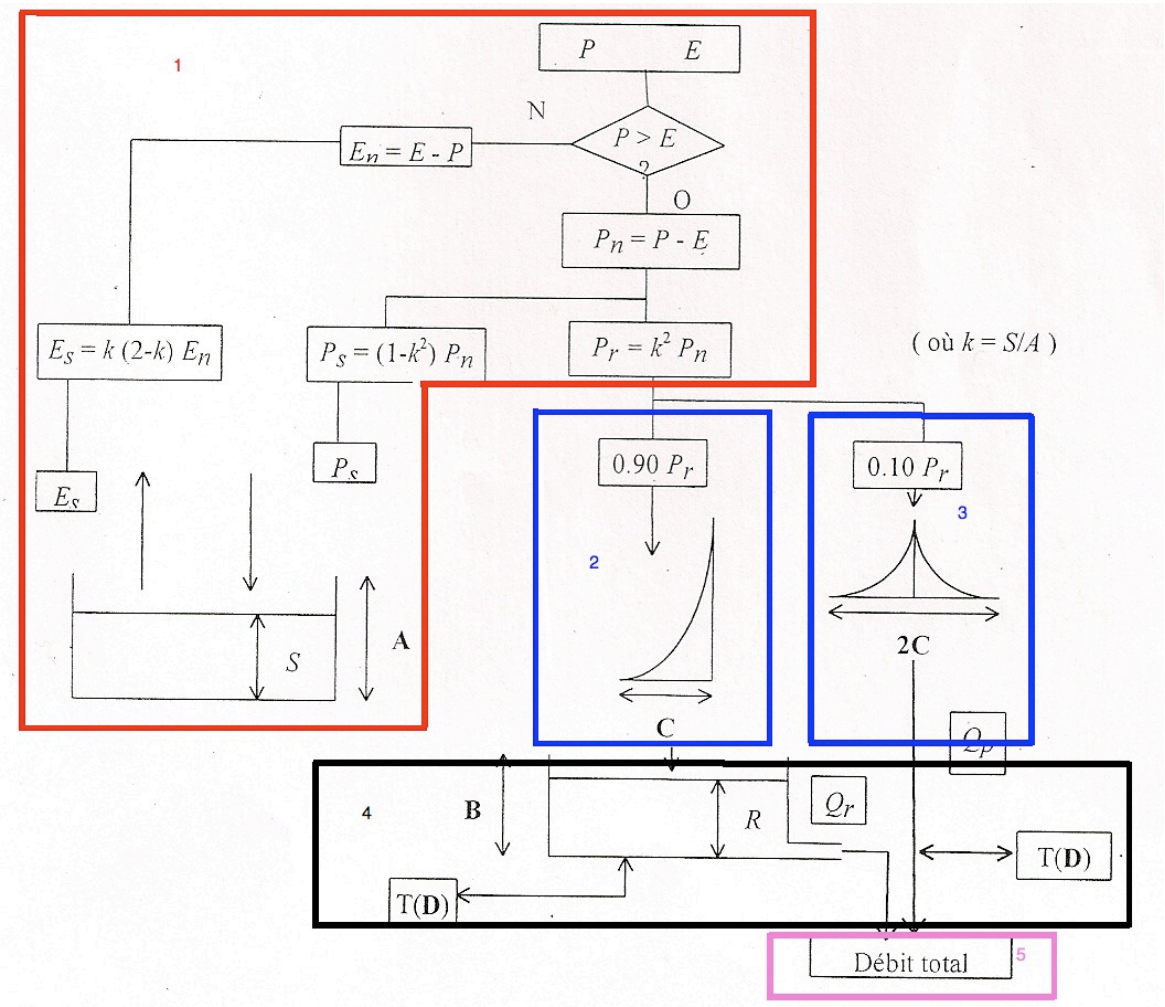

Figure 3 GR3J model

Figure 3 describes the GR3J model. We may point out that variables P and E represent respectively the precipitations and the potential evaporation. If the precipitations are greater than the evaporation the first transformation (Pn) is performed. Then the following two functions are computed:

- The production function: a part of the water goes through the soil reservoir which is defined by its capacity (noted down here A) and its real level S. S evolves according to the rain Pn and the evaporation En. The input (Ps) and output (Es) flows are taken into account when Pn and En are positive.
- The transfer function: the water that does not go into the soil reservoir represents the available water for runoff (called Pr in figure 3). This quantity of water is divided into two parts: the most important quantity is described in the left part in figure. The reader may notice in figure 3 that 90% is transformed by a 1-day unit hydrograph (let us call it SH1) while 10% is transformed by a second unit-hydrograph (let us called SH2). The first part, after routing by SH1 is given as input to a reservoir whose level is R and maximum capacity is B. This reservoir allows to obtain a first output called Qr in figure 3. The second hydrograph SH2 allows to generate a second output called Qp in figure 3.

  After passing through these two hydrographs the total of the two outputs (Qr and Qp) of the two hydrographs represents the total steam flow (called Debit Total in figure 3).

We have coded this scheme into DEVS models. We have been able to represent the behavior pointed out in figure 4 by the interconnection of the following five DEVS atomic models:

- The first atomic model (called ModelInterception) allows to model the part of the algorithm of figure 3 which corresponds to the red part and the number 1. The model receives precipitations as input, checks if the rain is greater than the evaporation and then sends the appropriate part to the soil reservoir and the other to the next model.
- ModelSH1 (in blue and noted 2 in figure 3) represents the first hydrograph; its take 90% of the water and send it to the reservoir model depending on a given formula.
- ModelSH2 (in blue and noted 3 in figure 3) represents the second hydrograph; the water quantity that is received as input is almost immediately sent on the output port.
- ModelRoutage (in black and noted 4 in figure 3) is the reservoir in which water goes after passing through SH1.
- ModelSomme (in pink and noted 5 in figure 3) allows to compute the sum of the output of the two hydrographs and therefore send on the output port the final output.

The interconnection of these five models allows to define a coupled model representing the GR3J model.

The snow model receives the temperatures and the precipitations as input to compute a result using a formula defined by a climatologist.

A complete description of these models can be found in [2] but we have to point out that the GR3J model presented in this paper was slightly modified in order to obtain better results.

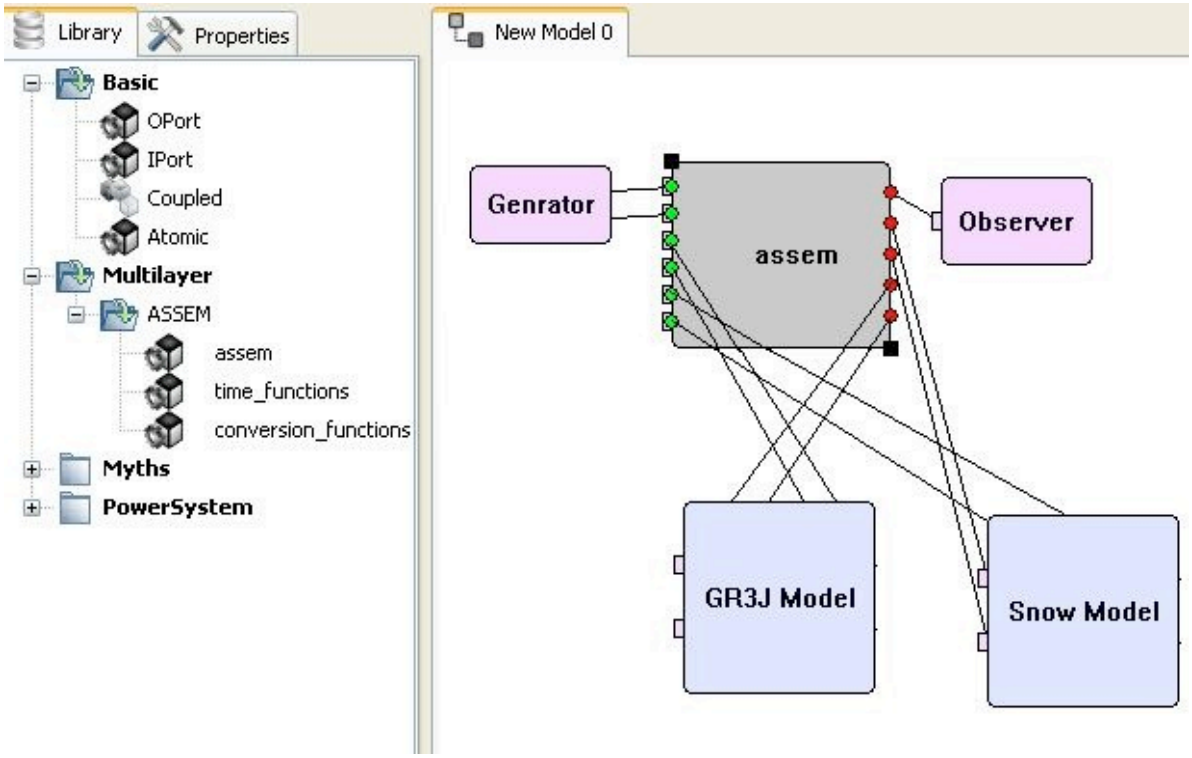

Figure 4 Multilayered architecture in DEVSIMPY

The next section describes how we have been able to interconnect these two models using a Python-DEVS and DEVSIMPY Assembly model implementation.

5.2 Python-DEVS and DEVSIMPY implementation

We describe in this sub-section how we have been able to interconnect the previously introduced two models using the assembly model. We present in figure 3 the coupled model allowing to interconnect the Assembly model (called ASSEM coupled model in figure 3) with the two models (GR3J and Snow atomic models).

In Figure 4 the left part of the window presents the domains available in DEVSIMPY. We add the multilayer domain, which contains the description of the assembly model, the generator, the observer, the GR3J Model and the Snow model. With a simple drag and drop we can add one of these models to the new Model, which is going to be created in order to perform simulations of the hydrologic behaviour of a catchment basin.

The hydrological and snow model are here considered as black box, so we are not able to change anything on it since specialists have separately defined them. We only know which kind of data (as well as its unit) is going to be sending on a given port. Owing to this information we have been able to create a kind of library which will help us in order to perform an automatic treatment of the data: let us call RoutageLibrary this library.

The conversion functions allowing the two models (GRJ3 model and snow model to communicate) are stored into a file and may be called by the stockage model when a data is sent on or sent to a given port. The stockage model receives a read or write order. Depending on this order it can execute two actions:

- Write order: the stockage model receives a value on one of its ports with a write order. Using the RoutageLibrary the unit of the considered data is obtained and the corresponding conversion function is invocated before storing the obtained data.
- Read order: the stockage model receives a read order on one of its ports. Using the RoutageLibrary we are able to find which kind of unit associated to the considered data and the corresponding conversion function is invocated.

5.3. Results

This part presents the results of this work. We have pointed out a comparison between real measurements taken by a specialist on a river chosen as example and the output of the simulation process. Figure 5 presents these results for a year and figure 6 between the year 1969 and 1972.

We can see that the obtained results fit well with the real measurements except for few parts of the two graphs. These differences are due to the GR3J model; this

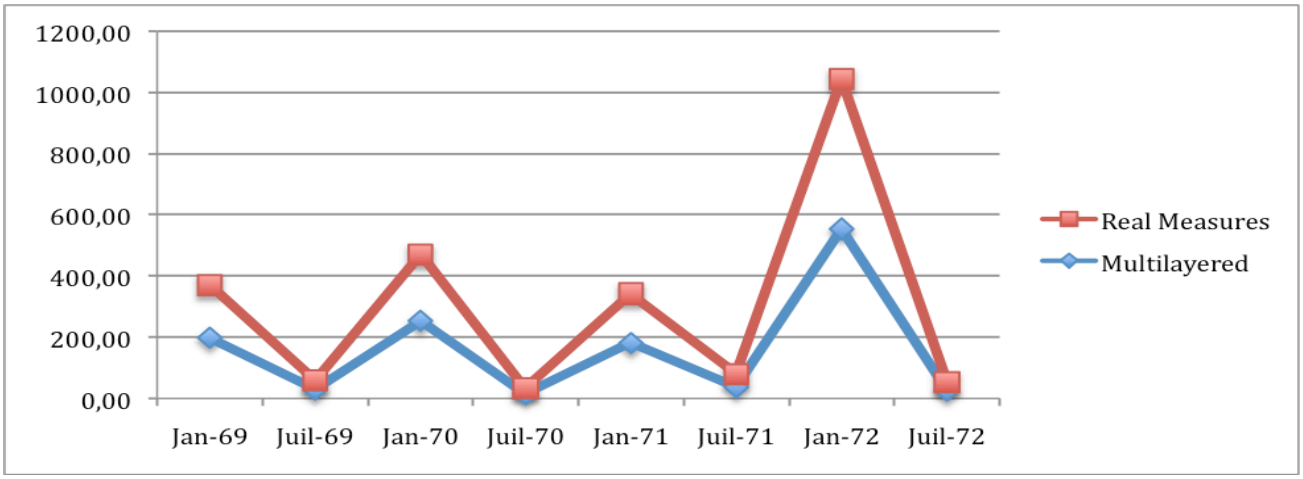

Figure 5 Comparision between real measures and results of our simulation for a year

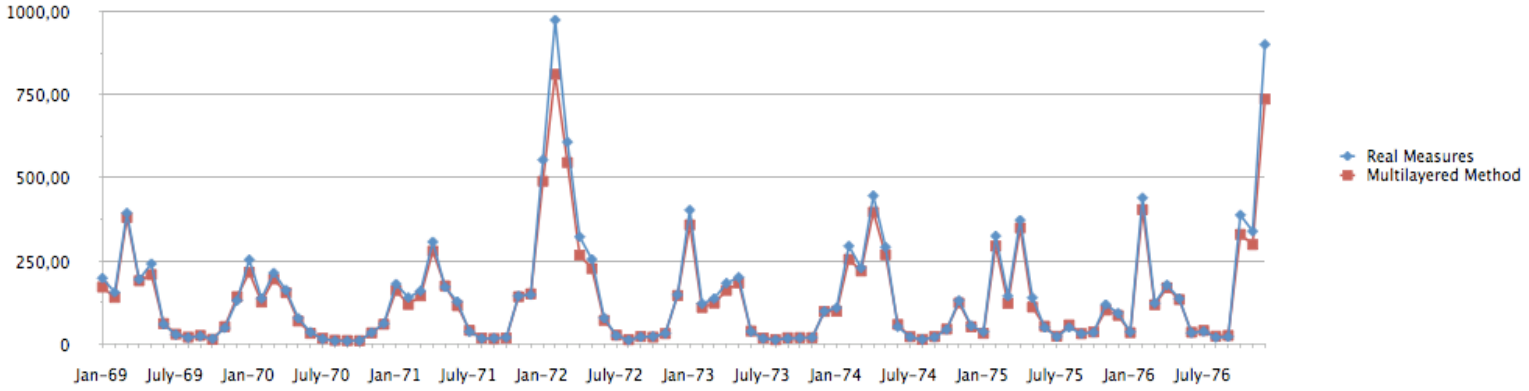

Figure 6 : Comparison between real measures and results of our simulation for several years

model is not suitable for the treatment of flood events. The highest peaks in the first graph (figure 5) correspond to flood periods.
We can see on the second figure (figure 6) that the differences correspond to the winter and spring period. We may point out that these periods correspond to a combination of rain periods with snowmelt periods which produces huge flood.
We finally can conclude that the obtained results are pretty good in comparison to the real measurement except when flood periods occur.

6. CONCLUSIONS AND PERSPECTIVES

We have presented here a validation of the multilayered formalism presented in [1]. We use a validation example dealing with the prediction of the behaviour of a catchment basin. This example perfectly fits to our problematic because of its complexity. By complexity we mean the use of different units of data used in order to describe the different models. The next validation example we envision will be the progression of a fire forest, which is a little more complex than the behaviour of a catchment basin.
We use Python-Devs and its extension DEVSIMPY for the implementation. We choose the python language for its simplicity but obviously not for its rapidity which is not here the objective.
The last part of this paper presented results obtained by running the combination of two separately defined models using the Assembly model. The obtained results show the efficiency of our approach. The next step will be the integration of a solution for dealing with temporal problems. The time management will be delegated to the Assembly model, so we will need to redefine the abstract simulator. In a first approach we plan to use a conservative time management method.

REFERENCES

[1] Broutin E, Bisgambiglia P, Santucci JF, “Simulation of heterogeneous DEVS models; application to the study of natural systems”. In proceeding of the Spring Simulation Conference 2009, San Diego CA.

[2] Broutin E, Bisgambiglia P, Santucci JF, « A PYTHON VALIDATION OF THE MULTILAYER DEVS THEORY: CASE OF A CATCHMENT BASIN, European Simulation and Modelling Conference, OCT 26-28, 2009 Holiday Inn, Leicester, UK, EUROPEAN SIMULATION AND MODELLING CONFERENCE 2009, Pages: 15-19

[3] http://moncs.cs.mcgill.ca/MSDL/reasearch/DEVS

[4] Liu B, Yao Y, Toma J, Wang H, “Implementation of Time Management In Runtime Infrastructure”.

[5] Praehofer H; Sametinge J, Stritzinger A, 2000. "Building Reusable Simulation Components," *presented at WebSIM2000, Web-Based Modelling and Simulation, San Diego, CA, USA.*

[6] Zacharewicz G, Frydman C, Giambiasi N, “G-DEVS/HLA Environment for Distributed Simulations of Workflows”, in: Simulation, n° 84, pp. 197-213, 2008

[7] Zeigler, B.P; Hall S.B; Sarjoughian H.S. 1999. "Exploiting HLA and DEVS To Promote Interoperability and Reuse in Lockheed's Corporate Environment," *Simulation*, vol. 73, number 4.

[8] Zeigler, B.P. 1975. “Theory of Modelling and Simulation” *Academic Press*.

[9] Zeigler, B.P. 1976. “Theory of Modeling and Simulation.” *New York, Wiley.*

[10] Zeigler, ,B.P. 1984. “Multifaceted Modelling and Discrete Event Simulation”. *London, Academic Press.*

[11] Zeigler, B.P. 1990. “Object-Oriented Simulation with Hierarchical, Modular Models”.

[12] Bolduc J.S. and Vangheluwe H, PythonDEVS: a modeling and simulation package for classical hierarchical DEVS. Technical Report, MSDL, McGill University, 2001

[13] Calvin, J.O, Weatherly R, “An Introduction To The High Level Architecture Runtime Infrastructure (RTI)”.

[14] Fujimoto R.M. “Time Management In The High Level Architecture”.

[15] Frederick Kuhl , Richard Weatherly , Judith Dahmann, Creating computer simulation systems: an introduction to the high level architecture, Prentice Hall PTR, Upper Saddle River, NJ, 1999_

[16] P. Benjamin,. Erraguntla JM, Delen D, Mayer R, "Simulation Modeling at Multiple Levels of Abstraction," presented at the 1998 Winter Simulation Conference, 1998

BIOGRAPHIES

Emilie Broutin is a PhD student in computer science in the University of Corsica. Her current research focuses on DEVS multilayered modeling and simulation.

Paul Bisgambiglia is a professor in Computer Sciences at the University of Corsica. He is responsible of the modeling and simulation team of the UMR CNRS 6134. His research activities concern the techniques of modeling and simulation of complex systems and the test of systems described at high level of abstraction. He makes his researches in the laboratory of the UMR CNRS 6134.

Jean-Francois Santucci is Full Professor in Computer Sciences at the University of Corsica since 1996. His main research interest is Modelling and Simulation of complex systems. He has been author or co-author of more than 100 papers published in international journals or conference proceedings. He has been the scientific manager of several research projects corresponding to European or industrial contracts. Furthermore he has been the advisor or co-advisor of more than 20 PhD students and since 1998 he has been involved in the organization of more than 10 international conferences. He is conducting newly interdisciplinary researches involving computer sciences, archaeology and anthropology: in the one hand he is performing researches in the archeaoastronomy field (investigating various aspects of cultural astronomy throughout Corsica and Algeria using tools issued from Computer Sciences) and on the other hand he is applying computer sciences approaches such as GIS (Geographic Information Systems) or DEVS (Discrete EVent System specification) to anthropology.