Discrete event simulation based on decentralised schedule and direct coupling

Paul-Antoine Bisgambiglia and Paul Antoine Bisgambiglia and Romain Franceschini
University of Corsica, UMR SPE 6134 CNRS, UMS Stella Mare 3460
TIC team
Campus Grimaldi, 20250 Corti
bisgambiglia@univ-corse.fr and bisgambi@univ-corse.fr

Keywords: Discrete event simulation, DEVS, Accelerated Simulation, Direct Coupling, Decentralised Schedule

Abstract

In this paper, we present an approach, based on DEVS formalism, aiming to optimize simulations time.
To do this, we propose to reduce the number of messages exchanged between components. If there are fewer messages to take into account, we believe saves processing time and thus accelerate simulations. We propose three changes from the classic DEVS: direct coupling, flat structure and local schedule. The goal is the decentralization of a number of tasks in order to make the simulators more autonomous, and relieve coordinators. All these changes were added by inheritance. Although, we proposed modifications of simulation algorithms, compatibility is still possible between classic DEVS models and decentralized DEVS models. The universal properties of DEVS are fully preserved.

1. INTRODUCTION

The study of problems linked to the behaviour of complex systems, ecosystems in particular, necessitates the development of specific tools. *Complex systems are characterized, not only by a large number of components, but also by the diversity of these components. For the analysis and design of such complex systems, it is not sufficient to study the diverse components in isolation, using the specific formalisms these components were modelled in. Rather, it is necessary to answer questions about properties (most notably behaviour) of the overall multi-formalism system* [1]. Most natural systems may be called complex. They may serve as a support for modelling and simulation studies. In a case of this type, simulation tools have become imperative for studying those phenomena.

At a theoretical level, two types of M&S approach oppose or complement one another. The first type of systems are discrete-time systems whose state is updated at each step, whereas the second type of systems are discrete-event systems that use a contiguous time base (state change only occurs at discrete points in time over this contiguous time base). The crucial benefit of simulation with discrete events is speed of execution owing to development dictated by events, avoiding processing in time stages. The formalism DEVS for Discrete EVent system Specification [2] is a formalism based on the development of time according to events. Simulation based on this formalism is therefore, generally speaking, faster than continuous simulation. DEVS formalism also allows the composition of models from components stored in libraries, thus avoiding the redevelopment of existing models. It is an open, flexible formalism with a great capacity for extension. Recent studies [3]–[6] have shown that DEVS formalism may be called multi-formalism because, due to its open nature, it allows the encapsulation of other modelling formalisms. For example, for the same heterogeneous system it is possible to use sub-systems modelled on different formalisms, differential equations, neural networks, continuous systems and loose systems. This capacity for opening and extension is very interesting, as the representation of the various entities which constitute a complex system can be accomplished by using the most appropriate formalism.

Although faster than a continuous interval simulation method, the DEVS approach is no less costly in terms of time when the number of models increases; many projects also deal with accelerating it from simulations. At the physical level, it is possible to exploit the power or the number of processors or computers, although the cost may be very high [7]–[9]. At software level, it is possible to improve the simulation algorithms by reducing their complexity. The third way, which we shall classify as a hybrid one because it links the first two methods, is aimed at developing the simulation algorithms to develop in order to parallelize the calculations [10].

In the DEVS formalism, the hierarchy between models suggests that all modifications of the states of a model involve sending a message, which goes as far as the highest level model. The number of messages is therefore proportional to the modifications of the system, to the number of models, to the changes in state and to the level of the hierarchy. In certain cases, such as models with highly advanced communication, this causes an explosion in the number of messages and a slowing of the simulation process.

Many studies have proposed approaches to accelerate simulations. We can cite: parallelization approach [7]–[12];

distribution [11]–[13]; and software approaches. These last approach allows to accelerate the algorithms by "destroying" the hierarchical structure, which corresponds to flattening the DEVS simulator [10], [12], [14], [15].

In this article, we propose software development which makes it possible to significantly improve simulation times by linking, to a flat simulation technique, a technique for managing the couplings between models (direct coupling), a new mode of managing messages and events and a major simplification of all the simulation algorithms. Currently, our approach is neither parallel nor distributed. We propose modifications of classical DEVS formalism.

In the first part, we present the general context of DEVS formalism. This description will allow us to gain a general impression of the modelling and simulation process with discrete events, as proposed by Zeigler, but especially better to understand the parts which we wish to modify. More especially, we stress the notion of message exchange, which is the most sensitive parameter in the calculation of simulation times. In the second part, we detail the structural modifications introduced to the "classic" DEVS environment, to accelerate simulation algorithms. In particular, we demonstrate that choosing to process messages as close as possible to the models allows us to limit their exchanges but especially to simplify the main algorithms. We provide a description of these algorithms for all the objects linked to the simulation. In particular, the last part allows us to use this "decentralised" simulation approach to study two systems and to present results obtained in a number of exchanged messages. We demonstrate that the major benefits make it possible to accelerate the simulations a great deal.

2. DEVS FORMALISM

In 1970's, Professor Zeigler introduced DEVS formalism [2]. It represents: (1) a complex system from an interconnected collection with more simple subsystems; (2) a separation between modeling and simulation part, simulation algorithm are automatically generated according to defined models. This formalism is open, flexible and offers a large extension capacity.

According to recent works [1], [3], it has been proved that DEVS formalism might be qualified as a multi-formalism thanks to its opening capacity, to its capacity to encapsulate others modeling formalisms. In one heterogeneous system, it is possible to use modeled subsystems from different formalisms, differentials equations, neuron networks, continuous systems.

DEVS formalism is based on the definition of two types of components: atomic models and coupled models.

Atomic model provides an autonomous description of the system behavior, defined by states, input/output functions and transition functions. The coupled model is a composition of atomic models and/or coupled models. It is modular and presents a hierarchical structure which enables the creation of complex models from basic models.

Let AM an atomic model, AM is defined by $< X; Y; S; t_a; \delta_{ext}; \lambda; \delta_{int} >$ where X: is the set of input events, is characterized by a couple (port, time, value), where the port means the input on which the event occurs, the time is the date of occurrence of the event, it is blank for internal events, and the value symbolizes the data from the event; Y: is the set of output events; S: is the set of partial or sequential states, which includes the state variables; $t_a: S \rightarrow T^{\infty}$: is the time advance function which is used to determine the lifespan of a state; $\delta_{ext}: QxX \rightarrow S$: is the external transition function which defines how an input event X changes a state of the system, where $Q = \{(s, t_e) \mid s \in S, t_e \in (T \cap [0, t_a(s)]\}$ is the set of total states, and t_e is the elapsed time since the last event, T is the total time of the simulation; $\lambda: S \rightarrow Y\iota$: is the output function where $Y\iota = Y \cup \{\iota\}$ and $\iota \notin Y$ is a silent event or an unobserved event. This function defines how a state of the system generates an output event, when the elapsed time reaches to the lifetime of the state; $\delta_{int}: S \rightarrow S$: is the internal transition function which defines how a state of the system changes internally, when the elapsed time reaches to the lifetime of the state.

Every state S is associated with a lifetime t_a, which is defined by the time advance function. When a model receives an input event X, the external transition function δ_{ext} is triggered. This function uses the input event, the current state and the time elapsed since the last event in order to determine what the next model state is. If no events occur before the time specified by the time advance function for that state, the model activates the output function λ (providing outputs Y), and changes to a new state determined by the internal transition function δ_{int}.

The DEVS models are executed by abstract simulators [2], [9], [16] that are independent from the models themselves. Consequently, separated concerns between models and implementations of simulation can be achieved and enhance the verification of each layer independently. DEVS is a popular method to simulate a variety of systems. However, since its introduction by Zeigler, significant efforts were taken to adapt this formalism to different fields and situations. The many proposed extensions proved its ability to extend and openness.

We use the DEVS formalism for modeling all types of systems. In some cases, depending on the system, the simulation can be very time consuming. There are many works that aim to accelerate the simulation. We can cite [17], [11], [12], [18]–[21]. Generally, these works propose modifications simulation algorithms to parallelize and / or distribute computations (outsource). We also present an approach to accelerate simulations without outsourcing the computations.

3. DECENTRALISED DEVS

In this paper, we propose an approach to reduce the number of messages exchanged between DEVS components. If there are fewer exchanged messages, we believe saves processing time and thus accelerate simulations. Our approach is based on the classic version of the DEVS formalism. Actually, it does not intend to be parallelized. We propose three changes from the classic DEVS: direct coupling, flat structure and local schedule. The aim of these modifications is to simplify the DEVS formalism, in order to make it more effective and faster. These changes were added by inheritance. Although we proposed modifications of simulation algorithms, compatibility is still possible between classical DEVS models and decentralized DEVS models. In addition, the universal properties of DEVS are fully met, such as closer under coupling. The three proposed changes are detailed in this section.

3.1. Modeling part: direct coupling

One cause of the amount of messages to manage in the DEVS formalism is the routing of messages. Routing generated by the hierarchical structure of the formalism. For example, a component C1 of level H2 cannot communicate directly with a component C2 of the same level (H2). This is the case for all components. Messages must always go back to father component, H1 or H0 level. This hierarchy is a source of communication too, but we will return in the following points. The fact of not being able to communicate directly with a component of the same level is a problem. We propose to add a list of coupling in atomic models. This list is declared as a state variable model. The models will thus knowledge of the components to which it is connected, and with which it must communicate. Without a higher-level component, it can directly send messages. This list of decentralized coupling has been added through inheritance. The first modification also allows us to remove the coordinators and propose flattening simulation architecture.

3.2. Simulation part: flattening architecture

In the hierarchical structure proposed in the DEVS formalism, we place the root at the top, and then come a coordinator (H0 level). To flatten the simulation architecture, we propose to delete all the coordinators below the H0 level. Other works already offer this mechanism [10], [12], usually in order to parallelize and distribute the simulation. Our goal is to make the standalone simulator by removing redundant communications. We still retain a coordinator called flat coordinator; it is positioned just below the root. It was him who gave the simulators execution order. It has a small schedule with an events number equal to the number of active simulators. For each simulator, the event with the smallest time trigger (tn) is inserted into the flat coordinator's schedule. This is the first event of each local schedule. The third modification is the addition of a local schedule in the simulators.

3.3. Simulation part: local schedule

We created through the mechanisms of inheritance, a new simulator which has its own schedule. This is a local or decentralized schedule. The purpose of this addition is to make the simulator more autonomous and to simplify the task of flat coordinator. The simulator retains the events it has to perform, and does only go up the first event of its schedule. This is the time at which it will reactivate.

All these changes are encapsulated in four new classes that inherit from classes' coordinator, simulator, atomic model and coupled model.

3.4. New classes

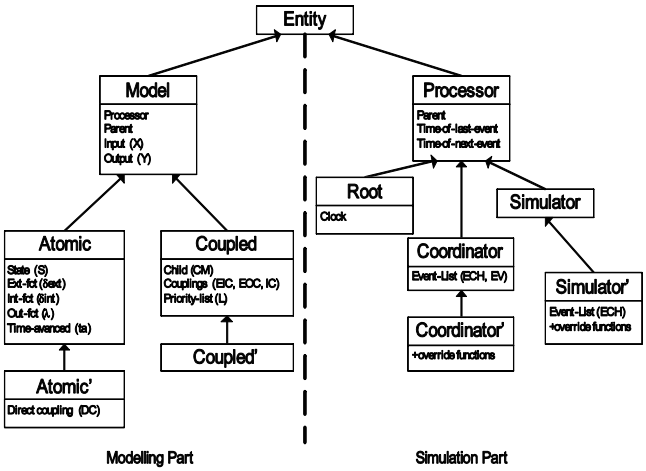

Figure 1. Correspondence between modelling and simulation in our approach

In the Simulator' class (see Fig. 1) modifications consisted of overloading the tn() function, in order to manage the messages directly without going through the coordinators, and in adding a local schedule. We propose using the time-of-last-event (tl) and especially time-of-next-event (tn) attributes to manage those messages in the coordinators. To conduct this decentralised message management, it is essential for the atomic model output ports and the input ports of the main coordinator to be able to identify the addressees without having to search for them in the coupling lists. A link notion is added to all those ports, thus making it possible directly to identify the "recipient" (model and port). It will then be possible to file the messages directly in the model schedules. This modification is applied in the atomic model class by overloading the Output (Y) objects and in the couplings model class by overloading Input (X) objects.
We are now going to present the behaviour of the various simulation functions which make up the core of the proposed abstract simulator [2], as well as the message send functions. All the simulation algorithms are based on the

result of the *tn()* function, which returns the time of the next event. In the case of a decentralised simulation, *tn()* returns the following:

- timeOfNextEvent for the Root processor
- $Min(\{tn()_c \mid c_{in} \in C_M\})$ for the root coupled model.
- Min(ta()+ct , time of the first ECH event) for the atomic models.

In the case of a distributed simulation, the number of schedules is equal to the number of atomic models. On the other hand, the number of messages managed is reduced and generally numbers about the same as the input ports. The management of those schedules is thus made much easier.

3.5. Algorithms

The general simulation algorithm is identical for the three types of simulation although, in the case of a decentralised simulation, the schedules for the root model are empty and the *tn()* function is only the minimum of the *tn()* of the sub-models. This modification has no effect on the simulation time and, for reasons of compatibility, searches in ECH (complete schedule) and EV (extraction of ECH at current time *ct*) are maintained in the "decentralised" simulation.

3.5.1. Root processor

The *Root* processor simulation algorithm defines the new current time of the simulation according to the data from the *inputData* (input data from the global model) and from the *Root tn()*. All the messages to be handled in ct time are extracted from the input schedule (or input file) and sent to the addressee via the *send()* function. This function will be detailed later, but for decentralised simulation, the messages arrive directly in the destination atomic model, since the input ports of the Root model recognise the addressee thanks to the *recipient* attribute. The simulation function (Algorithm 1.) makes it possible to perform the simulation in ct time, from the highest-level processor in the simulation tree.

```
1. function Simulate()
2. {  while (ct = Min(inputData.firstEventTime(),tn()))
3.  {       root.send(inputData.extractEventAtTime()) ;
4.          root.simulate() ;
5.  }
6. }
```

Algorithm 1: Algorithm of the Root processor's simulate function

3.5.2. Coordinator processor

In the case of a "decentralised" simulation, the messages only go up to the *root* if the message is a model output message. The couplings between the models are also no longer managed at coupled model level, but are handled by the atomic models. Each output port of an atomic model has the "recipient" attribute which gives it access to the model and to the message destination port. We are therefore able to place the output messages from the schedule directly in the addressee's schedule without having to search for it in the coupling functions of the coupled models.
The simulation function of the coupled model is in the following form:

```
1. function Simulate()
2. {  While (tn() == ct)
3.      ModelWithCurrentTime().processor.simuler() ;
       // You run the first model with tn=ct
4. }
```

Algorithm 2: Algorithm of the simulate function of the decentralised coordinator

3.5.3. Simulator processors

Decentralised simulation makes it possible to handle all the messages in ct time within the simulators without having to return control to the parent coordinator. The *while (tn()==ct)* loop (Algorithm 3. line 2.) is moved from the coordinators to the simulators.

```
1. function Simulate()
2. {  While (tn() == ct)
3.     if (ECH.firstEvent() != zero)
4.        model.deltaExt() ;    // handling external function
5.     else
6.     {  model.lambda() ;      // handling output function
7.        model.deltaInt() ;    // handling transition function
8.     }
9. }
```

Algorithm 3: Algorithm of decentralised simulators' simulate function

3.5.4. Handling messages

In the classic DEVS, messages are handled by two functions: send and insert. Send function making it possible to handle the messages on their arrival in a coordinator and to redirect them according to the couplings and to the priority list. Insert function uses message type and couplings to search for the destination model, to change the message into a result and to insert it in the corresponding schedule.

In the case of decentralised approach, message handling is simplified, since each port to which messages are sent recognises the "recipient" without having to go to the coupled model. It is also sufficient to modify the port attribute and possibly its type before inserting it into the destination model schedule (Algorithm 4.).

```
1. function Send(msg)
2. {  recipient = model.outputPort[msg.port].recipient;
3.    msg.port = recipient.port ;
```

```
4.   if (recipient is not coupledModel) msg.type = x ;
5.   recipient.model.insert(msg) ;
6. }
```

Algorithm 4: Algorithm of decentralised processors' send function

The *insert()* function deletes the old message (Algorithm 5. line 4.) if it exists and inserts the new one (line 5.). As previously indicated, this schedule is greatly reduced in size, as it is limited to the ex-messages of a model in ct time.

```
1. function Insert(msg)
2. {  positionMessage = this.Find(msg);
3.    if (positionMessage!= zero)
4.       this.removeAt(positionMessage) ;
5.    this.add(msg); }
6. }
```

Algorithm 5: Algorithm of the schedules' insert function in the case of decentralised models

To prove in concrete terms the value of these modifications and our algorithms, we have chosen to compare the three types of simulations: (1) the classic DEVS simulation in which the coupled models can be inserted inside other coupled models and create a hierarchical simulation tree, within which the messages are distributed. (2) A flat simulation in which the intermediary coupled models are deleted, which has the effect of placing all the atomic models at the same level under a root coupled model. (3) A decentralised simulation in which the structure is 'flat' and the messages are handled directly inside the atomic models.

4. EXAMPLE

The suggested approach makes it possible considerably to reduce the complexity of the simulation algorithms. It remains for us to demonstrate through some examples that this is expressed by a major reduction in the number of messages exchanged and in the execution time. We propose to present the results of GR system. GR corresponds to a more complex model developed by *Cemagref*, which makes it possible to represent a hydrological process and, in particular, to establish the link between the large volume of water hurled into a catchment reservoir and its flow at the outlet [22]–[24].

4.1. GR4J model

We now propose to compare the three types of simulation approach (classic, flat and decentralised) to provide more significant results.

The rural engineering models (GR [22]–[24]) are reliable, robust empirical models designed for annual, monthly and daily intervals, making it possible to achieve continuous simulations. They have numerous engineering and water resource management applications such as the proportioning and management of works, forecasting of water-level rises and low water levels, impact detection, etc. In order to function, those models only need continuous rain and potential evapotranspiration data, being capable of forming an average interannual curve. We are going to use the daily GR4J model with 4 parameters (Figure 2).

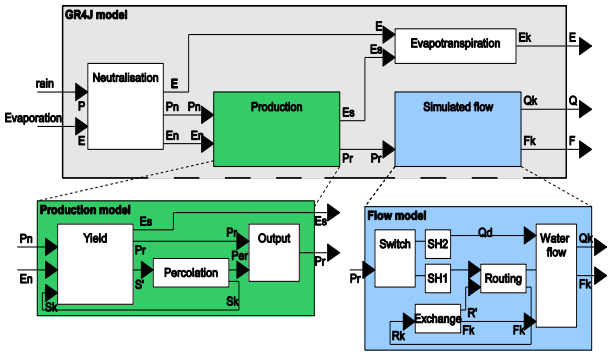

Figure 2. Architecture of GR4J model

The GR4J comprises four parameters to correspond to the catchment reservoir:

- X1: capacity of the production tank (in mm) in the percolation model;
- X2: coefficient of underground exchanges (mm) in the exchange model;
- X3: daily capacity of the routing tank (mm) in the routing model;
- X4: basic time of the unitary hydrograph in the SH1 and SH2 models.

We have studied the models:

- One model based on the classic DEVS formalism; its structure is identical to the figure showing the GR4J model. It is composed of 11 atomic models and three coupled models.
- A "flat" model composed of 11 atomic models and only one coupled model.
- A "decentralised" model, itself identical to the preceding model but composed of atomic models capable of managing the messages internally and links to the successors in the ports.

4.2. Framework used

The GR model has been implemented under the platform DEVSimPy. DEVSimPy framework allows a simple graphical interface to create and use DEVS models. It is designed from the PyDEVS architecture [25] but offers the possibility of using different simulation algorithms (classical, parallel, flat, decentralised) through strategy pattern (UML). It is a WxPython based environment for the simulation of complex systems. Its development is supported by the CNRS (National Center for Scientific Research) and the SPE research laboratory team.

The main goal of this framework is to facilitate the modeling of DEVS systems using the GUI library and the drag and drop approach. The interface is designed to help the implementation of DEVS model in form of blocks. The modeling approach of DEVSimPy is based on UML Software, and there is a separating between the GUI part and the implementation part of DEVS formalism.
With DEVSimPy we can: (1) describe a DEVS model and save or export it into a library; (2) edit the code of DEVS model to modify behaviors also during the simulation; (3) import existing library of models which allows the specific domain modeling (Power Systems, Fuzzy, Continuous ...); (4) automatically simulate the system and perform its analysis during the simulation.
Simulations results have been obtained from DEVSimPy.

4.3. Simulation results

Over a period of a year, we obtain the following for the three types of simulation (Figure 6):

- "classic": 17545 messages (series 1).
- "flat": 15543 messages (series 2).
- "decentralised": 7116 messages (series 3).

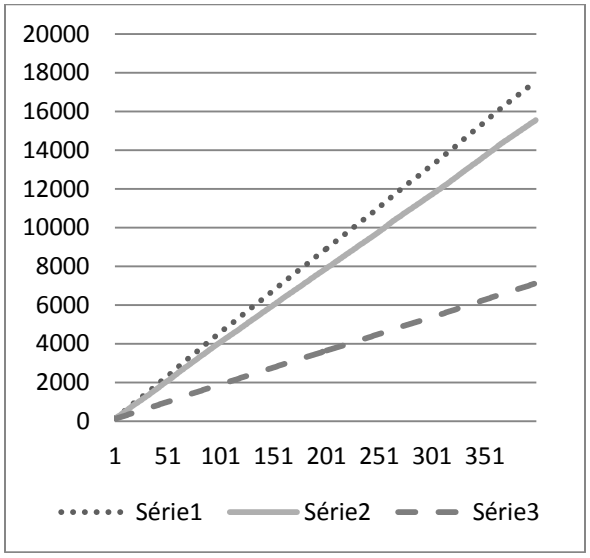

Figure 3. Message counting according to the simulation approaches

The difference in messages exchanged between flat simulation and classic simulation is approximately 2000 messages. This increase was predictable. In reality, it corresponds to a message cascade in the coupled sub-models which have been deleted. In the first case, we have two models coupled inside the GR4J model - there is, therefore, an extra level of communication (an extra send). In the coupled Production and Run-off model, we have three input ports and four output ports, making a maximum total of 7 messages on each loop, which is a maximum of 400x7 = 2800 messages.

It is easy to predict that, in the case of flattening, the increase in the number of messages is restricted by: Total Ports IN and OUT of the Coupled Model multiplied by the number of loops.

In the case of a decentralised simulation, the increase is initially obtained by deleting *-messages from the architecture. As an initial approximation, it may be said that each time an *-message is sent, a y-message is produced, which produces an upper limit of around 50%. What's more, an atomic model which retrieves control can handle all the messages in ct time, i.e. all the x-messages without returning control to the parent coordinator. We are also able to imagine increases above 50% in most cases of figures on a flat model.

Although those results are less significant, since they depend on the machine on which we have carried out the tests in the chosen language and the implementation of the simulation algorithms, we provide the results obtained on a personal computer with an application developed in C#.

We obtain simulation times (measured in ticks) proportional to the number of messages exchanged in the various approaches (**Figure**):

- "classic": 4970322 ticks.
- "flat": 4540248 ticks.
- "decentralised": 1910157 ticks.

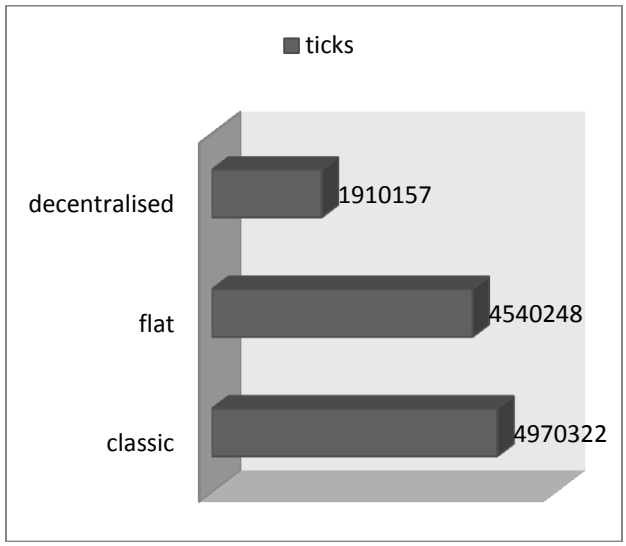

Figure 4: Simulation times (measured in ticks)

5. CONCLUSION AND PERSPECTIVES

In this article, we presented an approach that aims to reduce the number of exchanged messages in the DEVS formalism. For us, to reduce this exchange leads of execution time gain. In our approach reducing the number of messages goes through an overhaul of the role of simulators. Indeed, we propose three major changes

compared to classical DEVS formalism: direct coupling, flat structure and local schedule.

Direct coupling allows atomic models to send messages without to pass by their father (the high level component). We gain in reactivity, since it is no longer necessary to browse the couplings list of coupled model.

The flat structure eliminates Coordinators and reduces the traffic message between father and sons.

Local schedule allows simulators to self manage without a coordinator.

The goal is the decentralization of a number of tasks in order to make the simulators more autonomous, and relieve coordinators. All these modifications are taken into account using the mechanisms of object-oriented programming and overloading of some classic DEVS functions. Through this mechanism the universal property of DEVS are preserved, and it is possible to couple a classic model and a decentralized model.

The results obtained from the DEVSimPy framework are successful; the number of messages exchanged is divided by two.

Now we want to test our approach on other models, can be proposed a parallel version and compare our execution time with another DEVS framework [7], [26], [27] based on this approach [28], [29].

Biography

Use **Style Un-numbered Heading 1** for this heading. If space permits, include a brief biography of no more than 300 words for each author at the end of the article to give it greater impact and validity for the audience.

6. REFERENCES

[1] H. L. Vangheluwe, « DEVS as a Common Denominator for Multi-Formalism Hybrid Systems Modelling », in *Proceedings of ISCACS 2000*, 2000.

[2] B. P. Zeigler, H. Praehofer, et T. G. Kim, *Theory of Modeling and Simulation, Second Edition*. 2000.

[3] B. P. Zeigler, « DEVS Today - Recent Advances in Discrete Event-Based Information Technology », in *proc. of the 11th IEEE/ACM International Symposium on*, 2003.

[4] F. J. Barros, « Dynamic structure multiparadigm modeling and simulation », *ACM Transactions on Modeling and Computer Simulation*, vol. 13, n° 3, p. 259‑275, juill. 2003.

[5] G. Quesnel, R. Duboz, et É. 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, n° 4, p. 641‑653, avr. 2009.

[6] P.-A. Bisgambiglia, E. de Gentili, P. A. Bisgambiglia, et J.-F. Santucci, « Fuzz-iDEVS: Towards a fuzzy toolbox for discrete event systems », in *Proceedings of the SIMUTools'09, Rome (Italie)*, 2009.

[7] E. Glinsky et G. Wainer, « New Parallel Simulation Techniques of DEVS and Cell-DEVS in CD++ », in *proc. in Annual Simulation Symposium*, 2006, p. 244‑251.

[8] V. Balakrishnan, P. Frey, N. B. Abu-Ghazaleh, et P. A. Wilsey, « A framework for performance analysis of parallel discrete event simulators », in *Proceedings of the 1997 Winter Simulation Conference*, 1997.

[9] A. C. Chow et B. P. Zeigler, « Abstract Simulator for the Parallel DEVS Formalism », in *Proceedings of the Fifth Annual Conference on (AIS94)*, 1994.

[10] S. Jafer et G. Wainer, « Flattened Conservative Parallel Simulator for DEVS and CELL-DEVS », in *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 01*, 2009, p. 443‑448.

[11] Kihyung Kim, Wonseok Kang, Bong Sagong, et Hyungon Seo, « Efficient distributed simulation of hierarchical DEVS models: transforming model structure into a non-hierarchical one », 2000, p. 227‑233.

[12] G. Zacharewicz et M. E.-A. Hamri, « Flattening G-DEVS / HLA structure for Distributed Simulation of Workflows », in *Proceedings of AIS-CMS International modeling and simulation multiconference*, Buenos Aires, Argentine, 2007, p. 11‑16.

[13] Qi Liu, « Distributed Optimistic Simulation Of Devs And Cell-Devs Models With Pcd++ », 2006.

[14] S. Jafer et G. Wainer, *Global Lookahead Management (GLM) Protocol for Conservative DEVS Simulation*. .

[15] M. C. Lowry, P. J. Ashenden, et K. A. Hawick, « Distributed High-Performance Simulation using Time Warp and Java », DHPC-084, 2000.

[16] F. Barros, « Abstract simulators for the dsde formalism », in *Proceedings of WSC 1998*, 1998, p. 407‑412.

[17] G. Wainer, Q. Liu, et S. Jafer, « Parallel Simulation of DEVS and Cell-DEVS Models in PCD++ », in *Discrete-Event Modeling and Simulation*, vol. 20115630, G. Wainer et P. Mosterman, Éd. CRC Press, 2011, p. 223‑270.

[18] A. C. Chow et B. P. Zeigler, « Revised DEVS : A Parallel Hierarchical Modular Modeling Formalism », 2003.

[19] S. Jafer, Q. Liu, et G. A. Wainer, « Synchronization methods in parallel and distributed discrete-event simulation », *Simulation Modelling Practice and Theory*, vol. 30, p. 54‑73, 2013.

[20] A. Muzy et J. J. Nutaro, « Algorithms for efficient implementations of the DEVS & DSDEVS abstract simulators », in *Proceedings of the 1st Open International Conference on Modeling & Simulation*, France, 2005, p. 401‑4.

[21] F. Cicirelli, A. Furfaro, et L. Nigro, « Conflict management in PDEVS: an experience in modelling and simulation of time petri nets », in *Proceedings of the 2007 summer computer simulation conference*, San Diego, CA, USA, 2007, p. 349–356.

[22] Edijatno, N. De Oliveira Nascimento, X. Yang, Z. Makhlouf, et C. Michel, « GR3J: a daily watershed model with three free parameters », *Hydrological Sciences Journal*, vol. 44, n° 2, p. 263‑277, avr. 1999.

[23] C. Perrin, C. Michel, et V. Andréassian, « Improvement of a parsimonious model for streamflow simulation », *Journal of Hydrology*, vol. 279, n° 1‑4, p. 275‑289, août 2003.

[24] V. Andréassian, C. Perrin, et C. Michel, « Impact of imperfect potential evapotranspiration knowledge on the efficiency and parameters of watershed models », *Journal of Hydrology*, vol. 286, n° 1‑4, p. 19‑35, janv. 2004.

[25] H. L. Vangheluwe et J. S. Bolduc, « Pydevs ». McGill's, 2002.

[26] E. Kofman, M. Lapadula, et E. Pagliero, « PowerDEVS: A DEVS-Based Environment for Hybrid System Modeling and Simulation », 2003.

[27] E. Ramat et 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*, vol. 11, n° 1, p. 45‑55, mars 2003.

[28] E. Glinsky, G. Wainer, P. B. P. I, et C. Universitaria, « Performance Analysis Of Devs Environments », in *In Proceedings of AI Simulation and Planning*, 2002.

[29] G. Wainer, E. Glinsky, et M. Gutierrez-Alcaraz, « Studying performance of DEVS modeling and simulation environments using the DEVStone benchmark », *SIMULATION*, vol. 87, n° 7, p. 555‑580, janv. 2011.