Scientific Research Publishing

Open Journal of Modelling and Simulation, 2021, 9, 172-197
https://www.scirp.org/journal/ojmsi
ISSN Online: 2327-4026
ISSN Print: 2327-4018

DecPDEVS: New Simulation Algorithms to Improve Message Handling in PDEVS

Paul-Antoine Bisgambiglia, Paul Bisgambiglia

University of Corsica, CNRS UMR SPE, Campus Grimaldi 20250 Corti, Corsica, France
Email: bisgambiglia@univ-corse.fr

How to cite this paper: Bisgambiglia, P.-A. and Bisgambiglia, P. (2021) DecPDEVS: New Simulation Algorithms to Improve Message Handling in PDEVS. *Open Journal of Modelling and Simulation*, **9**, 172-197.
https://doi.org/10.4236/ojmsi.2021.92012

Received: March 13, 2021
Accepted: April 26, 2021
Published: April 29, 2021

Abstract

This work proposes a new simulation algorithm to improve message handling in discrete event formalism. We present an approach to minimize simulation execution time. To do this, we propose to reduce the number of exchanged messages between Parallel DEVS (PDEVS) components (simulators and coordinators). We propose three changes from PDEVS: direct coupling, flat structure and local schedule. The goal is the decentralisation of a number of tasks to make the simulators more autonomous and simplify the coordinators to achieve a greater speedup. We propose to compare the simulation results of several models to demonstrate the benefits of our approach.

Keywords

Simulation, PDEVS Formalism, Direct Coupling, Decentralised Schedule, Flat Structure

1. Introduction

Discrete event systems (DES) represent many technological and engineering systems [1]. They are driven by events, and these events have to be scheduled and sorted by their timestamps. The crucial benefit of simulation with discrete events is speed of execution owing to development dictated by events, avoiding processing in time stages. However, simulating a large number of models can be time-consuming because there are many events to handle. The DEVS formalism for Discrete Event system Specification [2] is a formalism based on the evolution of time according to events. DEVS and PDEVS [3] allow 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 [2] [4] [5] [6] [7] have shown that the DEVS formalism may be called multi-formalism because, due to its open nature, it allows the encapsu-

lation of other modelling formalisms. PDEVS [3] is a DEVS extension that eliminates serialisation constraints, provides a way of dealing with simultaneous events, allowing execution of models in parallel and distributed environments. Our contribution is based on PDEVS and simulation mechanisms will be presented in the next section.

Although faster than a continuous interval simulation method (Real-Time simulation excepted), the DEVS and PDEVS approaches are costlier in terms of time when the number of models increases. As the models' size and complexity in terms of links (couplings) increase, the longer calculation times become. In the PDEVS formalism, the hierarchy between models suggests that any model state modifications involve a message being sent, which goes up to the top-level model. Therefore, the number of messages is proportional to the number of modifications of the system, to the number of models, to the number of state changes and to the depth of the hierarchy. In certain cases, such as with highly interconnected models, this increases the number of messages and a slow-down of the simulation process. Many works also deal with accelerating simulations. At the hardware level, it is possible to exploit the power or the number of processors, GPU or computers, although the cost may be very high. At software level, it is possible to improve the simulation algorithms by reducing their complexity. The third way consists in combining the first two methods: it is aimed at implementing the simulation algorithms to develop in order to parallelize the calculations.

The question addressed in this work is: how to speed up the simulation. Many methods have been proposed to accelerate simulations. The main means used in the DEVS community to increase the speed or the number of models has been to exploit architectures with several processors or computer networks. The works cited above demonstrate that in the majority of studies for accelerating algorithms, the global model is divided into sub-model and each one is executed on distinct processor by an individual simulation process called logical process (LP). Based on PDEVS formalism, several tools [5] [8] [9] [10] support conservative [11] and optimistic [12] [13] methods of synchronization in parallel or distributed simulation environment. In all cases, considerable speedups are obtained with large-scale simulations applied to many application areas and the results generally depend on the number of logical processors. We propose to classify these approaches into four groups. 1) Software modifications, proposing more efficient simulation algorithms running on a single processor [14]. 2) Parallelization techniques, taking advantage of multi-processors CPUs provides simulation algorithms that are executed simultaneously on several processing units [10] [15]-[21]. 3) Distribution techniques, taking advantage of multi-computer architectures aim to propose simulation algorithms that are executed simultaneously on several computers through a network protocol [9] [22] [23] [24]. 4) Hybrids modifications, proposing parallel and distributed algorithms [12]. Our work belongs to the first of these groups.

To simulate the real system, it must be first partitioned into a maximum of

sub-systems, as independent from each other as possible. When there is a strong causal dependency between the components in a sub-system or when the execution is sequential for a set of components, one logical process is enough. Therefore, components that interact frequently should be placed in the same logical processor to reduce communication overhead in contrast to seldom interacting components, which should be split into different logical processes that are executed in a parallel or distributed way [25]. Hence, in many real-world applications, logical processes would host more than one component. Since the efficiency of parallel and distributed algorithms is largely limited not only by the inter-logical process protocols but also by the intra-process messaging for the components that reside on the same logical processor, we propose a technique to reduce this local communication. Message handling influences the execution time of the simulation.

The main proposition of this work is the implementation of a new simulation mechanism to reduce the number of exchanged messages within a logical process, in order to accelerate the local execution. A first proposal is made in [26]. This is achieved by simplifying the underlying simulation algorithms. The hierarchical architecture of the simulator is replaced with a flattened one to reduce communication among the components, associated with a technique (direct coupling) for managing the couplings between models and ensure that the outputs of the model are directly sent to the influenced. This technique allows reducing the overall number of propagated messages within a logical process. As the PDEVS formalism, components receive all simultaneous events scheduled for the same time. This approach (called DecPDEVS for Decentralised PDEVS) is complementary and can be exploited at the same time with the inter-logical processors' protocols to increase speed up (**Figure 1**).

In Section 2, we discuss the PDEVS formalism. This description introduces basic ideas about event-driven modelling and simulation techniques as proposed [2]. The third section is devoted to our proposal. Section 4 describes simulation algorithms by presenting the behaviour of all introduced classes in the abstract simulators. In Section 5, several examples had been tested to compare performances of our modifications with the original simulators and demonstrate that it

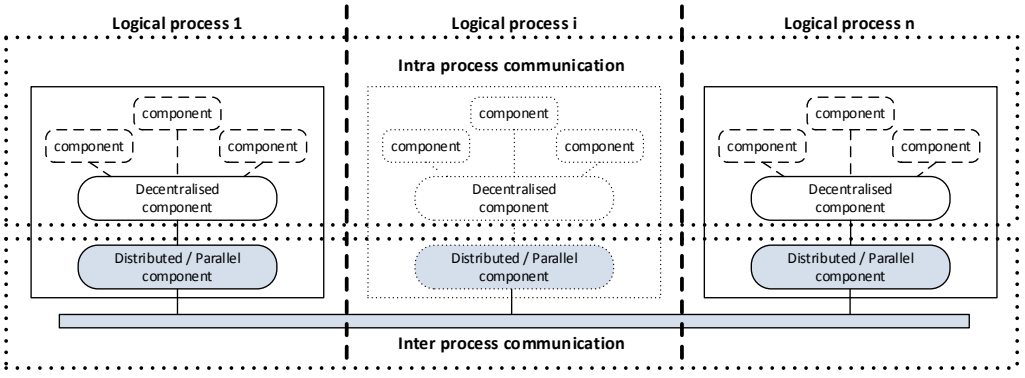

Figure 1. Decentralised architecture.

is possible to improve simulation performance. Section 6 concludes with a discussion on future research.

2. PDEVS Formalism

This formalism is based on the systems theory and the notion of the component. It allows the specification of complex discrete-event systems in a modular and hierarchical way. Major efforts have been made to adapt this formalism to various domains and situations [19] [27] [28] [29] [30] [31]. Within this vast sector, we are interested to improve the simulator.

2.1. Models

PDEVS is a modular formalism that permits the modelling of causal and deterministic systems. A PDEVS atomic model is based on continuous time, inputs, outputs, states and functions (output, transitions and states lifetime). More complex models are constructed by coupling several atomic models in a hierarchical way.

The atomic model may be considered as a time-based state machine. It makes it possible to describe systems' functional or behavioural aspects. The atomic model provides an independent description of the behaviour of a system, defined by its states and its functions. An atomic model is described by the following formula:

$$AM: <X;\ Y;\ S;\ t_a;\ \delta_{ext};\ \delta_{int};\ \delta_{con};\ \lambda> \quad (1)$$

with

- $X = \{(p_{in},\ v) | p_{in} \in$ Input ports, $v \in Xp_{in}\}$: the list of inputs events, each input being characterised by a tuple (port/value number);
- $Y = \{(p_{out},\ v) | p_{out} \in$ Output ports, $v \in Yp_{out}\}$: the list of outputs events, each output being characterised by a tuple (port/value number);
- S: the set of the system states or state variables;
- t_a: $S \longrightarrow R^+$: the time advance function marking states' lifetimes;
- δ_{ext}: $QxX^b \longrightarrow S$: the external transition function, where X^b is a set of bags over elements in X and:
  - $Q = \{(s,e) \mid s \in S, 0 \le e \le t_a(s)\}$;
  - e: is the time elapsed since the last transition.
- δ_{int}: $S \longrightarrow S$: the internal transition function.
- δ_{con}: $SxX^b \longrightarrow S$: the confluent transition to control the collision behaviour when an atomic model receives external event at the time of the internal transition $t_a(S)$;
- λ: $S \longrightarrow Y$: the output function.

At any given time, AM is in a state s for a lifetime period given by $t_a(s)$. When an external event occurs before $t_a(s)$ expires, a new state is given by the external transition function $\delta_{ext}(s,\ e,\ x^b)$. If the elapsed time e expires, the model outputs the value through $\lambda(s)$ and then evolves to a new state given by the internal transition function $\delta_{int}(s)$. When the atomic model receives events at the time of its

internal transition function $e = 0$ the confluent function $s' = \delta_{con}(s, 0, x^b)$ is executed instead of the internal or external function to determine the new state.

The DEVS formalism uses the notion of a description hierarchy, which permits the construction of models called "couplings", based on a collection of atomic models and/or couplings, and on three coupling relations. A PDEVS coupled model is modular and displays a hierarchical structure, which permits the creation of complex models based on atomic and/or coupled models. It is described by the formula:

$$CM: <X; Y; D; \{M_d\}; \{I_d\}; \{Z_{d,i}\}> \quad (2)$$

with: X: the set of input ports; Y: the set of output ports; D is the set of component references, M_d: the list of models that the coupled model CM is composed of; I_d is the set of influences of a model d for each $d \in D$, formed by a subset of $D \cup \{CM\} - \{d\}$.

- $Z_{d,i}$ is the translation function from d to i where $Z_{d,i}: X \longrightarrow X_i$ if $d = CM$, $Z_{d,i}: Y_d \longrightarrow Y$ if $i = CM$ and $Z_{d,i}: Y_d \longrightarrow X_i$ otherwise.

A coupled model describes the system structure and how the models are interconnected.

2.2. Simulation

In order to define the simulation semantics of DEVS components, Zeigler put forward the abstract simulator notion. The main benefit of this concept is the difference between the models and the simulator. At the level of this simulator (abstract), each simulation component corresponds to a modelling component. DEVS is one of the rare "formal" formalisms, which propose an implementation algorithm. In DEVS and PDEVS [2], a coupled model is composed of atomic and/or coupled models. The abstract simulator is composed of a root coordinator, coordinators and simulators. In fact, to achieve a simulation, a hierarchy of processors (root coordinator, coordinators and simulators) is constructed, equivalent to the hierarchy of models. A processor is associated with each model. Each processor carries out the simulation by performing the functions, which express the model's dynamics. The components are described as follows: the Root Coordinator, which represents the simulator, undertakes the general management of the simulation. It regulates the start and the end of the simulation process and manages the global clock; the Coordinators undertake the routing of events between the coupled models according to their couplings; the Simulators undertake the simulation of the atomic models by orchestrating the activation of its functions.

In an object-oriented based implementation of the formalism, the modelling part is based on the Model class, which contains a parent attribute to define the link in the description hierarchy and a processor attribute to form the link with the abstract simulator. On the simulation side, the abstract simulator classes inherit from the Processor class and possess all the time-of-last-event (tl) and time-of-next-event (tn) attributes, which allow synchronization of the events.

The coordinator class possesses also a *tl* and a *tn* attributes. The root-coordinator class has a clock attribute, which corresponds to the current time of the simulation.

2.3. Messages

Simulation is performed by sending specific messages between the various processors, as described in Figure 2. Each message contains information to identify the event type, the sender, the timestamp and the category. Two categories of messages are exchanged between processors. Content messages including external (x) and output (y) messages, used to transport data during the simulation, and synchronization messages including initialisation (i), collect (@), internal (*) and done (d) messages. The initialisation message marks the beginning of the simulation, the collect message triggers the output of the models, the internal message coordinates all models transitions and the done message is used for synchronization purposes, mainly for signalling the end of a task. Several abstracts simulators are proposed in [3] [8] [11].

i-messages are forwarded to all the processors at the beginning of a simulation run. When a simulator receives an i-message, it initializes the first state S of the atomic model, it computes $t_a(S)$ and initializes its times of next and last event. These values are sent to the parent coordinator. When a coordinator receives an i-message it initializes all the components in the M_d list and performs $tn = \text{Min}\{tn_d | d \in D\}$ and $tl = \text{Max}\{tl_d | d \in D\}$.

A coordinator always keeps different references to models. Several are partitioned into an imminent set (IMM) of models' candidates for the next internal transition function, and into an influenced set (INF) of models influenced by an IMM model.

When the coordinator receives an external **x-message**, it uses the $Z_{CM,d}$ (external input coupling) to determine the model and the destination. All these messages are stored in a bag for later use, when the ***-message** is received. When the coordinator receives outputs messages (**y-message**) from imminent components, it consults the translation function $Z_{i,d}$ to obtain the influenced and their

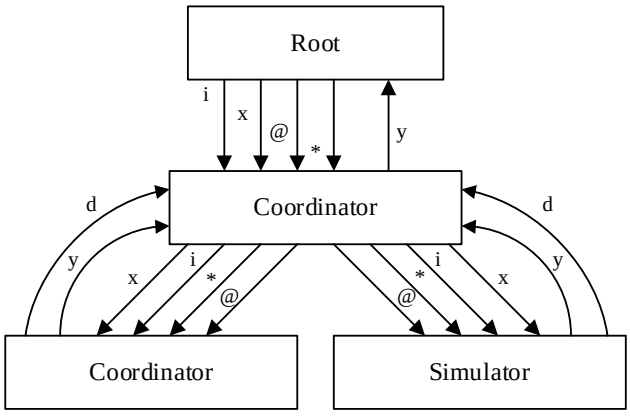

Figure 2. Simulation protocol.

respective input ports if $d \neq CM$ or the output ports if $d = CM$. In the first case, the output message is converted into input message ($x_d = Z_{i,d}(y_i)$), and a new x-message is sent to the influenced. In the second case, the output message y_{CM} is carrying to the parent coordinator ($y_{CM} = Z_{i,CM}(y_i)$). For the simulator, an **x-message** is simply stored in the bag and the simulator sends back a done message.

When the coordinator receives a collect **@-message**, it uses the imminent list (IMM) to propagate to the imminent processors a collect message, and waits for the receipt of all done messages before continuing. For a simulator, a collect message causes the execution of an output function ($y = \lambda(s)$), and the result is sent back to the parent coordinator. Both coordinators and simulators return a done message to their parent to signal the end of their collecting phase.

For the simulator, if an internal ***-message** occurs before the expiration of the current state lifetime of its associated atomic model and if the bag is not empty, the simulator triggers the external transition function, and the atomic model evolve to the resulting state of $\delta_{ext}(s, e, x^b)$. Note that all simultaneous events are passed to the model through the bag x^b. In case of collision between external and internal events, which occurs when the elapsed time is expired and the bag is not empty, the confluent function $s = \delta_{con}(s, x^b)$ is executed. If the elapsed time is expired and the bag is empty, an ***-message** causes the execution of the internal transition function $s = \delta_{int}(s)$. Finally, tl and tn are set to the current time and the next event ($tl + ta(s)$), and tn is sent back to the coordinator to report when the next internal transition should be executed. Finally, a d-message is returned to the parent.

When the coordinator receives an internal message ***-message**, it sends the external messages stored in the bag to the corresponding processors. All these receivers are added in a synchronize set and an internal message is sent to them. After all done messages are received back, the time of the next event and the time of the last event are calculated and a done message is sent to the parent. Another implementation might choose to route directly the events to the final influences during the (@, t) phase, the bag implementation of the coupled model can thus be omitted. The next section presents our proposal to improve simulators.

3. DecPDEVS: Decentralised PDEVS

We propose three new contributions from PDEVS: flat structure, direct coupling and local schedule to optimizing synchronization and output messages for intra logical processors communications. The aim of these modifications is to simplify the PDEVS simulator protocol, in order to make it more effective and faster, by eliminating some coordinators, output and internal synchronization messages. Although we proposed modifications of simulation algorithms, compatibility is still possible between PDEVS models and DecPDEVS models. In addition, the universal properties of DEVS are fully met, such as closure under coupling. The

three proposed changes are detailed in this section.

3.1. Flattening Architecture

Due to the hierarchical architecture of the DEVS formalism, the number of messages exchanged by the processors increase proportionally with the number of coordinators, and has a major impact on the elapsed time of a simulation. To reduce the number of these exchanges, all the coordinators, the top-most one excepted, are removed from the simulation tree and the couplings are reorganized between parent and children to preserve the same behaviour as that of the original. Other works already offer this mechanism, known as hierarchy flattening, [19] [22], usually in order to parallelize and distribute simulations. The flattened architecture that will be presented here is inspired from the various works, presented in Section 2.2, which are the attempts to improve the simulator to adopt a flattened structure. But we have not added any other type of coordinator as in [18] [32], we have one flat coordinator, called "decentralised coordinator", just positioned below the parallel/distributed coordinator (or the root for a single simulation). As shown in **Figure 1**, on each processor, one logical process encapsulates a decentralised coordinator. This coordinator assumes the intra processors communication.

Algorithm 1 describe the recursive "flatten()" procedure. First all the coupled models are scanned (lines 2 - 3), this execution replaces the original model by a flattened one. Next the information about atomic models are added to the local variables D, $\{M_d\}$ and $\{I_d\}$ (line 5 - 6) and in the sets of influences $\{I_d\}$ all the references to the child coupled models, except for the current model (self) are deleted (lines 7 - 8). Internals translations between two atomics models in the children are copied into the local Z function (lines 9 - 10). For the other translations,

Algorithm 1. Flattening algorithm.

```
1. procedure flattening()
2.   foreach cm in D when m_cm is a coupled model
3.         m_cm.flattening()
4.   foreach cm in D when m_cm is a coupled model
5.         D = D ∪ m_cm.D − cm and {M_d} = {M_d} ∪ {m_cm.M_d} − m_cm
6.         {I_d} = ({I_d}− I_cm) ∪ ({m_cm.I_d} − m_cm.I_cm)
7.         foreach d in D
8.               remove cm in I_d
9.         foreach transition z_d,i in {cm.Z_d,i} when (d ≠ cm and i ≠ cm)
10.              add transition (z_d to z_i) to {Z_d,i}
11.  foreach transition t_d to t_i in {Z_d,i} when (d or i are coupled model) //t∈{x, y}
12.        if (M_i is a coupled model) and (M_i ≠ self)
13.              x_j = m_i.Z_i,j(Z_d,i(t_d))
14.              replace (Z_d,i (t_d) = t_i) by a translation (Z_d,j (t_d) = x_j) into {Z_d,i}
15.              if (d is not a coupled model) add j to {I_d}
16.        if (M_d is a coupled model) and (M_d ≠ self)
17.              y_j is a output as M_d.Z_j,d(y_j) = t_d
18.              replace (Z_d,i (t_d) = t_i) by a translation (Z_j,j (y_j) = t_i) into {Z_d,i}
19.              add i to {I_j}
20. end flattening
```

we use the transitive property of the Z function, to compute the new route to the final receiver (lines 12 - 15) and the real sender (lines 16 - 19).

3.2. Direct Coupling for Message Routing

The simulators are the child processors of the logical processor, which represent the atomic components of the PDEVS models. Two kinds of messages are processed in the logical process; internal messages which have an impact to the states of all atomic models $S = \bigcup_{d \in D} S_d$ of the decentralised coordinator (in the local logical process) and external messages which have an impact to states of atomic models assigned to others LPs. When an internal event occurs, the simulators are not able to communicate directly their outputs to their receivers; all the messages must be forwarded up to the decentralised coordinator. Consequently, internal messages increase communication among participating processors. For better understanding, recall that when a simulator wants to send a message to another simulator in the same LP, it first sends an output y-message to the coordinator and a d-message for synchronization. Then, the coordinator consults the Z translation function to find the destination and the input port, translates the y-message into an external x-message and sends it upward to the influenced. The simulator stores events in a bag and returns a d-message to notify that the message is processed.

To reduce this communication overhead, we propose to add a list of couplings to the atomic models. The atomic model (am) will have knowledge of its influencees ($I = I_{ma}$), and the translation function ($C_i = Z_{ma,i}$) from the sender to the receiver ($i \in I$). These additional variables are added through inheritance, and initialised during the initialization phase. This structure is one of the benefits of our approach, because it is associated to the direct coupling notion, as vaguely proposed in [14] and called "implicit link", to directly connect two atomic models and allow them to communicate without travel through the parent coordinator. In addition, this concept eliminates unnecessary internal synchronization messages (*i.e.*, d-messages), and avoiding unnecessary event routing messages (*i.e.*, y-messages and x-messages). As specified in [Chow, 1994] routing the events directly from the topmost coordinator to the final simulator is equivalent as a classic implementation and yields the same simulation results, for the internal event the result is similar.

3.3. Sending Message

During the collect phase, *i.e.* when the coordinator spreads a @-message among its children, each simulator propagates the outputs of its associated model to their recipients and in turn, may receive inputs from other simulators. In the original simulation protocol, a simulator collects the outputs and propagates y-messages to its parent and signal the end of its collect phase by propagating a d-message for synchronization purposes. For each y-message, the coordinator has to consult the couplings to dispatch those messages to the proper recipients using

x-messages, or again y-messages for those intended to simulators located in other LPs. All messages intended for simulators in the same LP are passing unnecessarily by the top-most coordinator. Using the direct coupling notion presented in Section 3.2, the output events are directly sent by the simulator by propagating x-messages to recipient simulators using the local translation function C_i. At the end of the overall collecting phase, all simultaneous messages properly are in the bags of the simulators, ready to be processed during the internal phase (*-message).

The post function for the simulator is presented in Algorithm 2. If the output of the result is intended to its parent, the result (values in the output bag y^b) is stored in the output bag of the decentralised coordinator (line 4). In the other case, the result is inserted in the input bag of the atomic model (line 6). For the coordinator (Algorithm 3) input events are directly inserted in the corresponding bag.

3.4. New Classes

We present our structure modifications in Figure 3 which reflects the decentralised organisation. The modelling and simulation approach follows the definition of the PDEVS abstract simulator [Zeigler, 2000]. The decentralised abstract simulator is separated in two class hierarchies, one for the model and the other for the simulator. There is a one-to-one correspondence between class models and class simulators for the decentralised components, where the D-Coordinator and the D-Simulator represent the behaviour of the models. The simulation and the advancement of time are managed by the inter-process in parallel or distributed situation or by a root simulator in case of a mono-processor architecture. For a logical process a network processor (parallel, distributed or root) running on different machines determines when the intra processor simulation phase has to be started and the final execution time, corresponding to the "lookahead" in the PDEVS conservative algorithms [Jafer, 2010, Wainer, 2002, Zacharewicz, 2008 and 2010]. The D-coordinator class is responsible for routing the messages

Algorithm 2. Post algorithm for a simulator.

```
1. function post(y^b)
2.   foreach y in y^b
3.         if i destination of C_i(y) is a parent
4.               add y = C_i(y) to the bag   m_i.y_i^b
5.         else
6.               add x = C_i(y) to the bag of   m_i.x_i^b
7. end post
```

Algorithm 3. Post algorithm for a coordinator.

```
1. function post(x^b)
2.   foreach x in x^b
3.         add x = Z_cm,i(x) to the bag of   m_i.x_i^b
4. end post
```

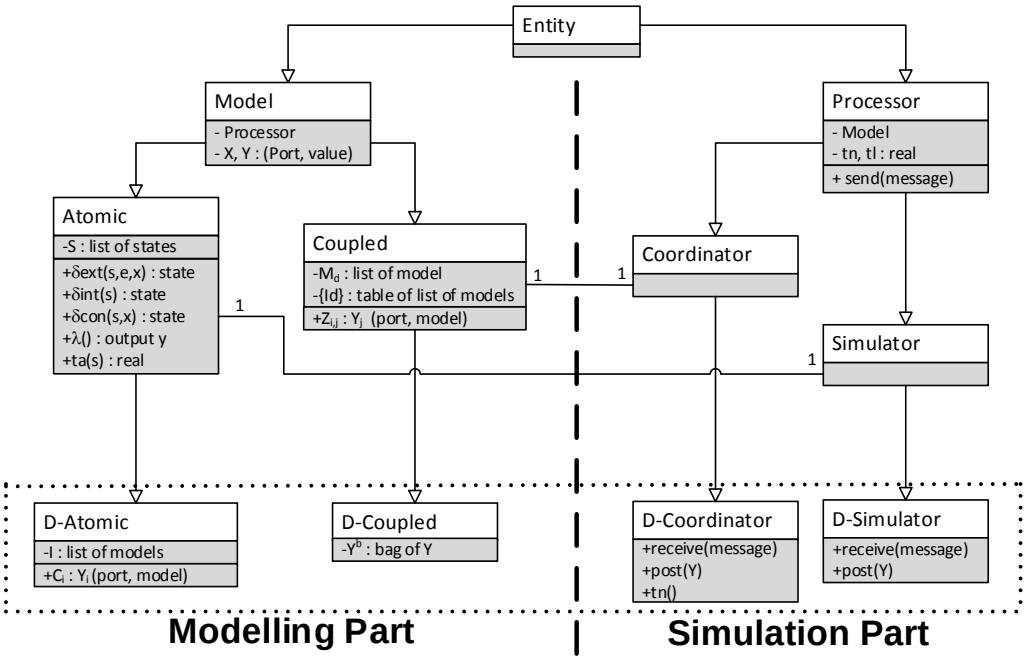

Figure 3. Decentralised abstract simulator.

among inter and intra processors parts. The intra-simulation start with the internal messages sent from the network processor to the D-coordinator and continues until the timestamps of the events reaches the final execution time or until there are no more events with a timestamp less than or equal to the final execution time. All output messages are sent out by the D-coordinator at the end of the simulation.

In the D-Coordinator class modifications consisted in overloading the *tn*() function, in order to handle the time of next event directly without using the *tn* value of the done simulators. We propose using the time-of-last-event (*tl*) and especially time-of-next-event (*tn*) attributes in the coordinator, thus the *tn* computation that involves a comparison of processor parameters is fast efficient.

To conduct this decentralised message management, it is essential that the atomic model output ports and the input ports of the main coordinator are able to identify the addressees without using the translation function Z. 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 bags. This modification is applied in the atomic model class by the state I and the function $\{C_i\}$. This notion is called direct coupling or implicit link. In the simulation part every processor input and output messages are treated as basics event and inserted directly into the corresponding bag by the function "post".

We are now going to present the behaviour of the various simulation functions, which make up the core of the proposed abstract simulator. The simulation is message driven and managed by three specifics functions "send", "post" and "receive" implemented by each simulator. These functions translate content

and synchronization messages between models using direct couplings and execute the behaviour of the atomic models. Direct coupling represents a major modification in the modelling part, the translation function Z is used by the coordinator only for routing the input events ($Z_{cm,i}$|cm is a coupled model).

We proposed algorithms for purely conservative simulation. The simulator only executes events when it can guarantee that other simulators will not send events with a smaller timestamp than that of the current event.

4. Decentralised Algorithms

In this section, we describe decentralised algorithms discussed. To obtain a comparative study between PDEVS, flat and decentralised simulators for intra-processors simulation, we use only one logical processor and a root coordinator directly connected to the decentralised component. The root coordinator is a special processor, which is responsible to read/write the input/output events to the environment and starting the simulation. The root is driven by a special function called "global simulation" which synchronizes the global time simulation, routes down the input events, the collect and the internal messages (Algorithm 4).

4.1. Root Processor

Initialization message start the global simulation (line 2) to every processor in the simulation tree. The Root processor defines the new current time of the simulation according to the data from the inputData (input data is a list of input events) and to the *tn* of the topmost coordinator in each simulation cycle (lines 5 and 11).

All the messages to be handled at current time are extracted from the input data and sent to the topmost coordinator via the *send*() function (lines 5 - 7). Notice that a version of this function with a list of messages is used to reduce the number of x-messages sent. The collect message can be sent to compute the output for the current time of the simulation (line 8). Finally, the smallest timestamp (*lookahead*) of the next event the root can schedule in the future is computed (line 9), and send to the topmost coordinator with the internal message

Algorithm 4. Global simulation.

```
1. function global_simulation()
2.     send ("i", 0, Ø) to the topmost coordinator
3.     t = min(tn of the topmost coordinator, time of first event in inputData or ∞;)
4.     while (t ≠∞)
5.         foreach event ev in inputData with time = t
6.             extract ev from inputData
7.             send("x", t, ev) to the topmost coordinator
8.         send("@", t, Ø) to the topmost coordinator
9.         lookahead = time of first event in inputData or ∞;
10.        send("*", t, lookahead) to the topmost coordinator
11.        t = Min(tn of the topmost coordinator, time of first event in inputData)
12. end global_simulation
```

(line 10) to finish the current simulation cycle. This value guarantees that no other events containing a smaller timestamp will be generated. When the internal message is received the coordinator can safety executes all transitions and outputs functions from the current time t to the time of the next external event schedule in the *inputData*.

A receive function (Algorithm 5) is activated when a message is sent by a decentralised coordinator. If the time of the message is smaller than the time of the last event a causality error is detected (line 2). When an output message is received, the event of the message is stored in timestamp order in the output data base (line 4) and the time of last event is update (line 5).

4.2. Decentralised Coordinator

A decentralised coordinator contains simulators running the atomic models. To perform the simulation in the decentralised coordinator the sets of IMM and INF are determined to know which models are candidates for a collect @-message and which models are candidates for an internal *-message (Algorithm 6). INF is updated when an event is stored in the input bag of the atomic model, and IMM is computed using the *tn* attributes of the processors.

- $\text{INF} = \left\{ m_d \cdot x^b \neq \varnothing \mid m_d \left\{ M_d \right\} \right\}$
- $\text{IMM} = \left\{ m_d \cdot tn = \text{current time} \mid m_d \left\{ M_d \right\} \right\}$

When an initialization message is received the time of last event is set to zero (line 3 Algorithm 6) and the message is forwarded to all its children to perform the initialization phase (lines 4 - 5). Upon receiving a collect message with a timestamp equal to the *tn* value (line 6), the coordinator sends a collect message to the imminent processors (lines 8 - 9) for output execution. Notice that the results are directly assigned to the output bag of the coordinator and done-messages are not expected to be received from the children. When an internal message is received (line 10), the decentralised coordinator is in charge of processing intra processor simulation from the current time to the value (corresponding to the lookahead) received from the parent coordinator. All the events scheduled during this period (present it the IMM or INF sets) are executed, and the local time of the intra-processor is advanced.

Upon the first step, input messages stored in the local bag are distributed to the recipients (line 11). During simulation cycles (line 12), the decentralised coordinator sends internal messages to all its children ready for a transition (lines 13 - 14). After processing the collect messages output events are routed to the

Algorithm 5. Receive function of the root processor.

```
1. function receive(message msg)
2.     if (msg.time < tl) return causality error
3.     if msg is a y-message
4.             save the event of the msg in the outputData
5.             tl = time of the message msg
6. end receive
```

Algorithm 6. Receive function of the coordinator.

```
1. function receive(message msg)
2.   when receive an initialization i-message with t = 0
3.         tl = 0
4.         foreach d in D
5.               send ("i", t, Ø) to m_d
6.   when receive a collect @-message with t = tn
7.         tl = t
8.         foreach m in IMM
9.               send ("@", t, Ø) to the child m
10.  when receive an internal *-message with tl <= t <= tn
11.        post (x^b)
12.        while (IMM ∪ INF ≠ Ø)
13.              foreach m in IMM ∪ INF
14.                    send ("*", t, Ø) to the child m
15.              send ("y", t, y^b) to the parent
16.              tl = t
17.              tn = Minimum {m_d.tn|m_d ∈ {M_d}
18.              if (tn < msg.value)
19.                    t = tn
20.                    foreach m in IMM
21.                          send ("@", t, Ø) to the child m
22.  else if tn < t < tl return causality error
23. end receive
```

root coordinator to be stored in the environment (line 15), and the next scheduled transition time are computed (lines 16 - 17). In case of one or more processors are ready to process output function before the lookahead time expiration (line 18), the local time of the intra process time is updated (line 19), and a new collect message is sent to all processors with minimum *tn* (lines 20 - 21). If the time of the message is not within the range of *tl* and *tn* an causality error occurs (line 22).

4.3. Simulator Processors

Decentralised simulation handles all messages at the current time, executes the transition functions and generates the outputs without having to give back control to the parent coordinator. When an initialization message is received (line 2), the states of the model and the time of the last event are assigned (lines 3 - 5). Using the time advance function, the time of the next scheduled output event is obtained (line 5). A collect message (line 6) executes the output function (line 7) and deposits the result in the influenced bag (line 8). This event will be executed later, when an internal message will be sent to the model. An internal message (line 9) triggers one of the three transition functions based on the time (*t*) of the event, the time of the next transition event and the events stored in the bag. If the message arrives before the time of the next event (line 10), the external function is executed (line 12) with a new value for the elapsed time (line 11). If the timestamp of the message is equal to *tn* and the bag is empty (line 13) the internal transition function is executed (line 14). However (line 15) a conflict arises when the timestamp is equal to *tn* and the bag is not empty and the confluent

function is executed (line 16). At the end of the transition, the last and the next transition times are updated (line 17 - 18). If the time of the message is not within the range of *tl* and *tn* a causality error occurs (line 19) (Algorithm 7).

4.4. Example: CPU Models

To prove in concrete terms the value of these modifications and our algorithms, we have chosen to compare the three types of simulations:

- The PDEVS 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
- 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. Our flat version resulting from the literature [Jafer, and Wainer, 2009; Zacharewicz, 2010].
- A decentralised simulation in which the structure is "flat" and the messages are handled directly inside the atomic models

To illustrate the differences between PDEVS, flat-PDEVS and DecPDEVS simulation, a simple coupled model example (Figure 4) based on a queue and a processor used for the DEVS formalism modelling and simulation [Zeigler, 2000]. Queue is a buffer that stored incoming jobs and sends one of them to the processor when it is free. The processor simulates the job's execution delay or quantum, and remains busy until the processing is finished. If the execution delay is reach, the job is placed to the "out" port and the new status free is sent on the "done" port, otherwise the job is sent through the "done" port to the queue. The environment using a random function to generate: Jobs, arrival time and delay.

Figure 5 shows the result set from simulations based on PDEVS algorithms,

Algorithm 7. Receive function of the simulator.

```
1. function receive(message msg)
2.     when receive an initialization i-message with t = 0
3.         initialize system's states S
4.         tl = 0
5.         tn = ta(s)
6.     when receive a collect @-message with t = tn
7.         y = λ(s)
8.         post (y)
9.     when receive an internal *-message with tl <= t <= tn
10.        if (t ≠ tn)
11.            e = t - tl
12.            s = δ_ext(s, e, x^b)
13.        else if (t = tn) and (x^b = Ø)
14.            s = δ_int(s)
15.        else if (t = tn) and (x^b ≠ Ø)
16.            s = δ_con(s, x^b)
17.        tl = t
18.        tn = tl + ta(s)
19.    else if tn < t < tl return causality error
20. end receive
```

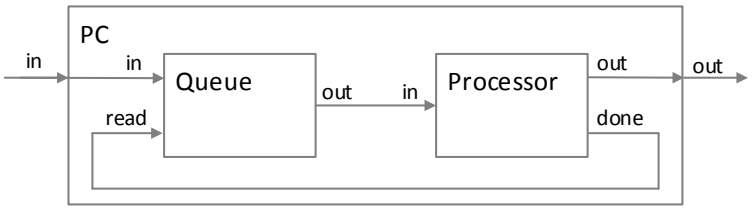

Figure 4. PC (Personal Computer) coupled model.

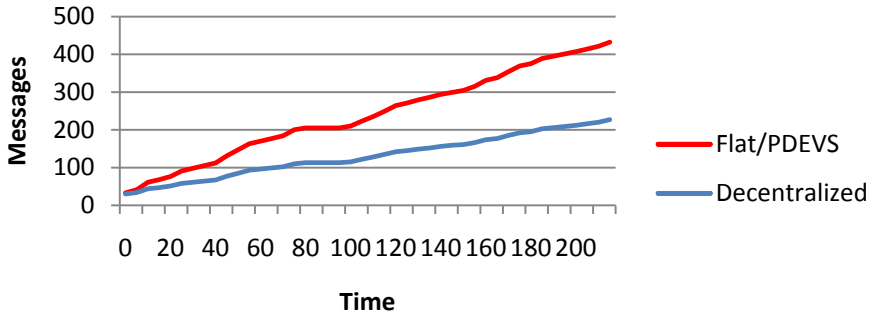

Figure 5. message counting according to PDEVS and decentralised approaches.

flat approach and on decentralised approach. As there is no intermediate coupled models in the hierarchy, PDEVS using the hierarchical and flattened architecture give the same results. Compared to the decentralised approach, the number of exchanged messages is significantly lowered, even for such a simple model. In this simulation, the gain is around 50% compared to a standard PDEVS simulation. In the beginning of the simulation, the improvement is about 15%. Then, in the middle of the simulation, we observe a 40% and 50% improvement. We can see that even for a very simple system without hierarchy, there is much less messages to process, which conducts to a better use of hardware resources.

In the next section, we will propose other examples to prove that the time saving is important regardless of the application type.

5. Applications

There are two ways to prove the efficiency of algorithms: 1) use a benchmark; 2) or, use known models. In [26], we demonstrated the interest of our approach from a DEVStone benchmark [33] and we propose a first set of test obtained with this benchmark. These preliminary results show an improvement in terms of exchanged messages. The improvement is around 70% compared to the classical approach and 40% compared to the flattened approach. In this section, we propose to realize new tests using real case models. Here we offer an in-depth study from real models. We will present two examples based on more complex models. We will compare the performance of three approaches (PDEVS, flat-PDEVS, and DecPDEVS) applied to two different systems. The first corresponds to a more complex model developed by Cemagref[1], which allows to represent a hydrological process and, in particular, to establish the link between the large

[1]https://www.enseignementsup-recherche.gouv.fr/cid49674/irstea.html

volume of water hurled into a catchment reservoir and its flow at the outlet (GR4J model) [34] [35] [36]. The second example corresponds to the Acceptable Safety Distance (ASD) model [37]. ASD is a model that allows the determination of safety distances for wild land fires. This analytical model is based on radiative heating and on fire spreading model [38]. In this section, we will present our test platform, and the method used to count the messages and evaluate the simulation time. Then, we describe the results obtained for the GR model, and finally those of the ASD model.

5.1. Materials and Methods

The presented results were obtained from models implemented with the DEVS-Ruby API [39]. It is a library (API) that allows formal specifications of DEVS and PDEVS (Parallel DEVS) models. It provides an internal Domain-Specific Language (DSL), which can be extended to meet domain specific vocabulary of modellers. We added other simulation algorithms in our code, and have included a message counter in the various simulation components.

The test environment is based on an Intel(R) Core (TM) i5-3360M CPU @ 2.80 GHz (3MB L2 cache), 16 GB (2 x DDR3 - 1600 MHz) of RAM, a Toshiba MK5061GS hard drive, running on Ubuntu 14.04 (64 bit). The software used for the benchmarking are: DEVS-Ruby 0.6 using the official Ruby VM (version 2.1.2).

To count the messages, we have added a counter in the code of the simulator. This counter is incremented by the simulators and the coordinators whenever the *send*() function is executed. It returns the total number of messages sent to each step of the simulation. In our simulation results, we distinguish two notions of time. The simulation time: the simulated time that is configured before to running the simulation (time), and its duration real or effective, given in CPU time (ticks).

5.2. First Application: GR4J Model

The rural engineering models called GR models [34] [35] [36] 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 four parameters (**Figure 6**). 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;

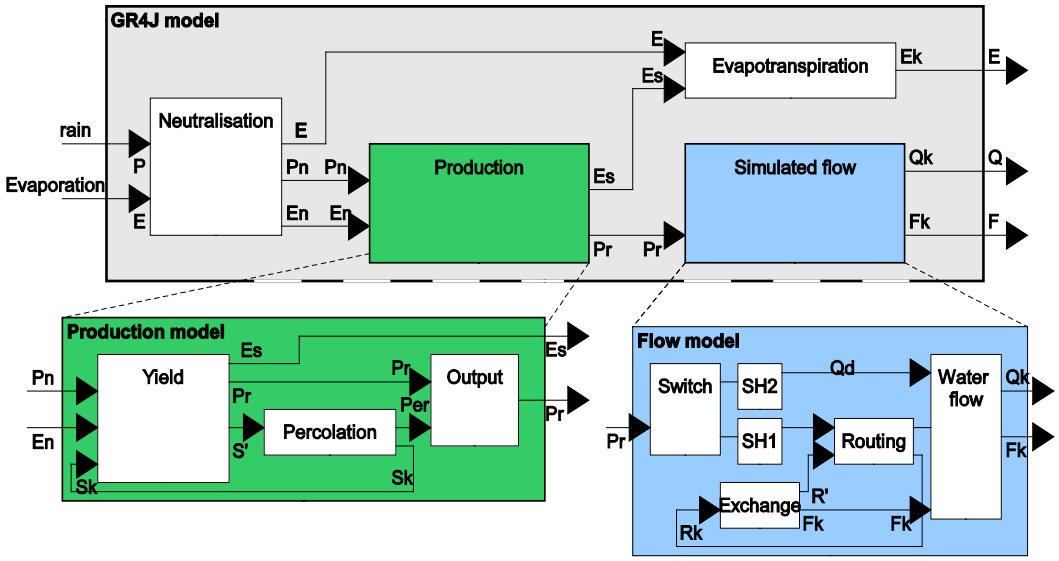

Figure 6. PDEVS structure of GR4J models.

- X4: basic time of the unitary hydrograph in the SH1 and SH2 models.

We have studied the models:

- One model based on the PDEVS 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.

Over a period of a year, we obtain the following for the three types of simulation (Figure 7):

- "PDEVS": 17,545 messages.
- "flat": 15,543 messages.
- "decentralised": 7116 messages.

The difference in messages exchanged between flat simulation and standard (PDEVS) 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 $400 \times 7 = 2800$ 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

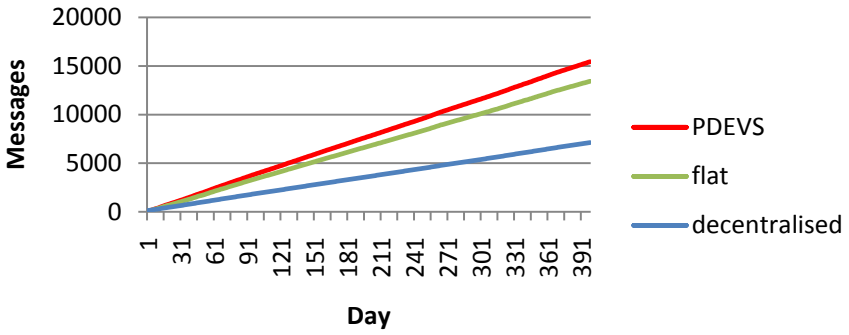

Figure 7. Message counting according to the simulation approaches.

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 at current time, *i.e.* all the x-messages without returning control to the parent coordinator. We are also able to imagine increases above 40% 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 following empirical results.

We obtain simulation times (measured in ticks) proportional to the number of messages exchanged in the various approaches (**Figure 8**):

- "PDEVS": 4,970,322 ticks = 497.0322 ms
- "flat": 4,540,248 ticks.
- "decentralised": 1,910,157 ticks.

In this example, we can see that if the model is more complex, with more coupling, our method allows obtaining even better results. The resource in time is improved by about 50% compared to the flat approach.

5.3. ASD Model

In [37], an application to calculate a safety distance is presented; it is called ASD for Acceptable Safety Distance. The role of this system is to calculate a safety distance for the prevention of forest fires. This distance is used to realise a firewall by vegetation clearing. We have represented this model in the PDEVS formalism for application that needs to allow fire fighters to calculate a safe distance on the ground from a touch pad. This model is very complex because it must solve nonlinear differential equations using the fixed-point method. This method requires setting up an iterative scheme that induces the creation of the loop and causes a lot of data exchange. The model is based on the following characteristics; all equations are given in [40]:

- ASD is the Acceptable Safety Distance (m), it represents the safety distance facing the fire;
- γ flame tilt angle, the inclination angle between the flame and the ground normal;

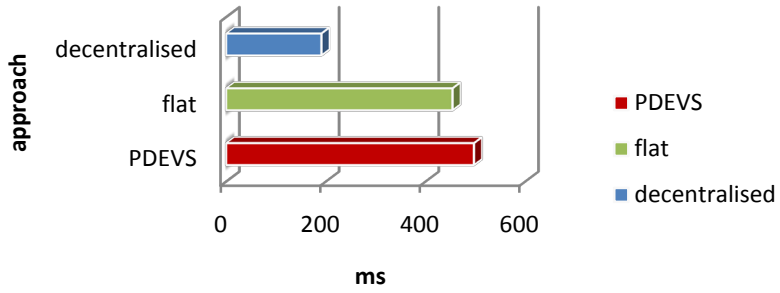

Figure 8. Simulation times (measured in ticks).

- R is the speed of the flame front;
- Lf is the length of the flame to the ground;
- R_0 is the flat velocity without wind;
- u_0 is the rate of climb of gas
- A is the ratio energy radiated/necessary
- Hf is the flame height

Modelling this system using the PDEVS formalism gives a coupled model composed of 21 models, there are eight models of first level: 4 atomic models and four coupled models, and 13 models of second level. DEVS models are described in [29] [37].

The four models coupled are CM_SD, CM_GV, CM_HR and CM_CF.

- CM_SD gives the ASD;
- CM_GV gives the rate of climb of gas, and the energy ratio;
- CM_HR gives the flame height;
- CM_CF gives the flame temperature and the flat velocity.

These four models are composed of 13 atomic models, model of second level. The four first-level atomic models are generator, parameter r0, AM_LF, and AM_S.

- generator initializes all components by reading the input data from a file or from a web service;
- parameter r0 calculates a ratio surface/volume (depending on ground);
- AM_LF gives the length of the flame;
- AM_S calculates the flame inclination and the flame front velocity.

Figure 9 shows the link between parameters or equations and their inclusion in DEVS models. This figure allows us to see the couplings and dependence or interdependence of a parameter with the others. These models have been implemented using our API (DEVS-Ruby [39]). The given results do not show the model output but shows the comparative study between the three different simulation algorithms. To do this study, we counted messages and measured simulations times.

Figure 10 shows the evolution curves in terms of number of exchanged messages. We can see that for the decentralised approach, the number of exchanged messages is much lower than for the other approaches. The gain is approximately 60% to 70%. This reduction allows a great saving of time, we can see in

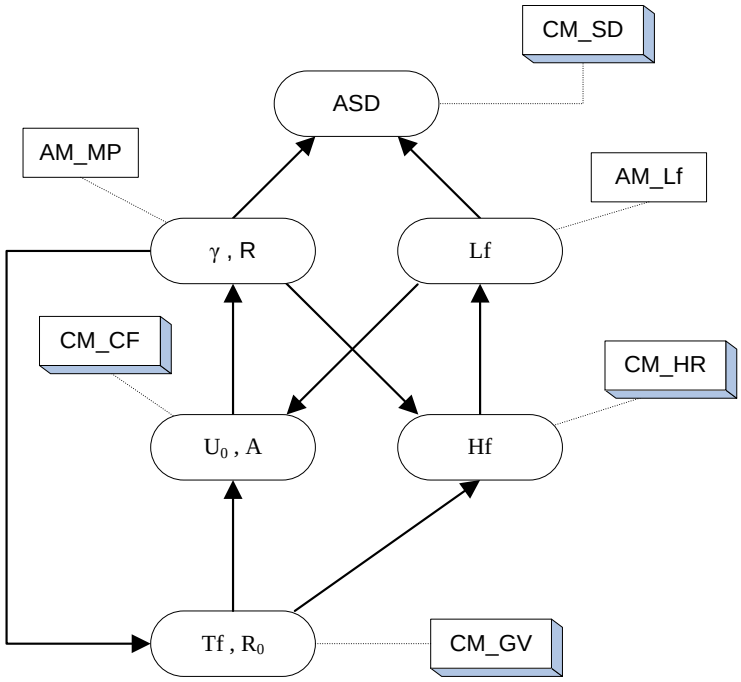

Figure 9. Link between the ASD model parameters.

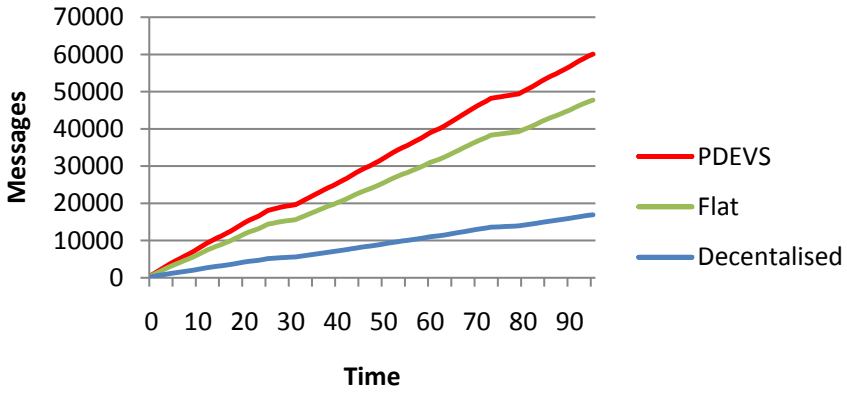

Figure 10. Message counting according to the simulation approaches.

Figure 11, which shows the CPU time (ticks) allocated for each simulation. We obtain simulation times, measured in ticks, based on the various approaches:

- "PDEVS": 30,550,550 ticks = 3055.0550 ms
- "flat": 11,030,118 ticks.
- "decentralised": 10,001 ticks.

In this last example, our previous observations are confirmed. More the system is coupled more the approach is effective. We improved message handling. Consequently, we decreased the number of sent messages consistently.

5.4. Discussion

For each simulation, we calculated the average gain in number of messages exchanged by approach (PDEVS, flat, decentralised). We find that for a simple

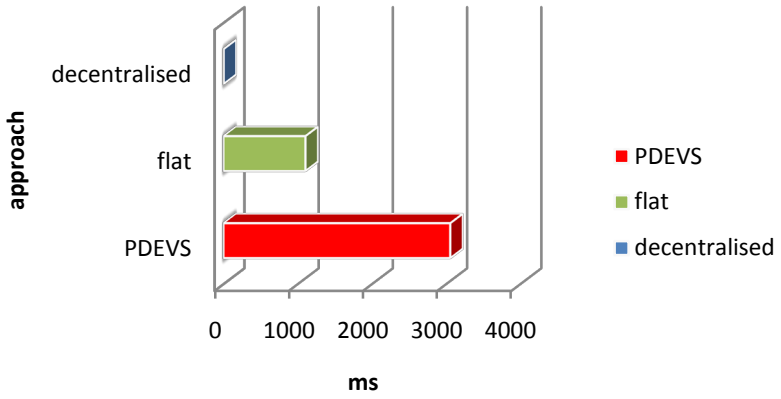

Figure 11. Simulation times (measured in ticks).

model, that is to say, composed of few models, the gain is of the order of forty percent (40%). As more models are present, the results are more and more encouraging. For the ASD model, which is composed of many models, the gain is approximately of 70% compared to the standard algorithm, and 60% compared to the flattened approach.

As shown in our results, this reduction in the number of exchanged messages leads to an acceleration of simulation time. For the CPU model, there are 72 outputs. For the GR4J model, there are 398 outputs. For the ASD model, there are 95 outputs. Our results are therefore fairly significant because they are verified for three model types and a DEVS benchmark (DEVStone [26]). We also note, and it is important, that the most significant improvement is seen with the ASD model, which is a discrete time model. We wish to confirm on other models of the same type, but we believe that our approach is optimal for discrete time systems.

Our approach has a major advantage when the modelled systems are complicated. For example, models with many ports and receiving many messages at the same time. More the number of (in/out)-ports is important, and therefore the number of messages sent or received also, more the benefit is important. Our approach provides an improvement for all models tested, we will accelerate the simulation time by about 50%. This acceleration will be cascaded on all models. For relatively simple models with few couplings, our approach is unattractive. Another limitation is that it is not yet possible to couple in a logical process a classic DEVS model with a decentralised model.

6. Conclusions

In this article, we propose the implementation of new simulation algorithms to reduce the number of exchanged messages in PDEVS to accelerate its execution. Our algorithms aim to reduce the number of messages exchanged between components during intra process communication. If there are fewer exchanged messages, we save processing time and thus accelerate the simulations. This approach is based on three changes from PDEVS. Although we proposed simula-

tion algorithm modifications, compatibility between PDEVS and DecPDEVS models is preserved (closure under coupling).

Our acceleration process in computer science is fairly traditional. It is mainly based on the relocalisation of information: blending list and local schedule in order to reduce the sizes of the data structures and to accelerate the search functions. On the modelling side, we added a list of couplings in the atomic model and the latter knows the models to which it is connected and thus, can send them a message directly (direct coupling), without the intermediary of the parent coordinator. We are able to transform an y-message directly into an x-message or another y-message in case of output from the local process to the remote process. On the simulation side, we used a local schedule (bag) to manage all the messages locally, without having to communicate with the parent coordinator. All these modifications are considered using the mechanisms of object-oriented programming and overloading of some PDEVS functions.

In terms of messages exchanged, both for simple or more complex models, our results are highly satisfactory, as we have halved the number of messages. Depending on the complexity of the model, we can earn between 50% and 70%.

It would now be interesting to implement our approach in another modelling environment than [39] [41] [42] in order to confirm our results and to prove the genericity of our approach.

Acknowledgements

The present work was supported in part by the French Ministry of Research, the Corsican Region and the CNRS.

The authors are very grateful to the anonymous referees and area editor for their comments and suggestions that greatly help to improve the quality of the manuscript.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Cassandrass, C.G. and Lafortune, S. (1999) Introduction to Discrete Event Systems. Springer, Berlin. https://doi.org/10.1007/978-1-4757-4070-7

[2] Zeigler, B.P., Muzy, A. and Kofman, E. (2018) Theory of Modelling and Simulation: Discrete Event & Iterative System Computational Foundations. Academic Press, Cambridge.

[3] Chow, A.C. and Zeigler, B.P. (2003) Revised DEVS : A Parallel Hierarchical Modular Modeling Formalism.

[4] Barros, F.J. (2003) Dynamic Structure Multiparadigm Modeling and Simulation. *ACM Transactions on Modeling and Computer Simulation*, **13**, 259-275. https://doi.org/10.1145/937332.937335

[5] Quesnel, G., Duboz, R. and Ramat, É. (2009) The Virtual Laboratory Environ-

ment—An Operational Framework for Multi-Modelling, Simulation and Analysis of Complex Dynamical Systems. *Simulation Modelling Practice and Theory*, **17**, 641-653. https://doi.org/10.1016/j.simpat.2008.11.003

[6] Zeigler, B.P. (2003) DEVS Today—Recent Advances in Discrete Event-Based Information Technology. *The Modeling, Analysis and Simulation of Computer Telecommunications Systems* 2003, *MASCOTS* 2003, Orlando, 12-15 October 2003, 148-161.

[7] Bisgambiglia, P.-A., Innocenti, E. and Bisgambiglia, P. (2018) Fuzz-iDEVS: An Approach to Model Imprecisions in Discrete Event Simulation. *Journal of Intelligent & Fuzzy Systems*, **34**, 2143-2157. https://doi.org/10.3233/JIFS-171020

[8] Himmelspach, J., Ewald, R., Leye, S. and Uhrmacher, A.M. (2007) Parallel and Distributed Simulation of Parallel DEVS Models. *Proceedings of the* 2007 *Spring Simulation Multiconference*, Volume 2, San Diego, 249-256. http://dl.acm.org/citation.cfm?id=1404680.1404720

[9] Kim, K., Kang, W., Sagong, B. and Seo, H. (2000) Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One. 33*rd Annual Simulation Symposium*, Washington DC, 16-20 April 2000, 227-233.

[10] Wainer, G., Liu, Q. and Jafer, S. (2011) Parallel Simulation of DEVS and Cell-DEVS Models in PCD++. In: Wainer, G. and Mosterman, P., Eds., *Discrete-Event Modeling and Simulation*, CRC Press, Boca Raton, 223-270. https://doi.org/10.1201/b10412-12

[11] Jafer, S., Liu, Q. and Wainer, G.A. (2013) Synchronization Methods in Parallel and Distributed Discrete-Event Simulation. *Simulation Modelling Practice and Theory*, **30**, 54-73. https://doi.org/10.1016/j.simpat.2012.08.003

[12] Kim, K.H., Seong, Y.R., Kim, T.G. and Park, K.H. (1995) Distributed Optimistic Simulation of Hierarchical DEVS Models. *Proceedings of the* 1995 *Summer Simulation Conference*, Ottawa, 24-26 July 1995, 32-37.

[13] Lee, H., Zeigler, B.P. and Kim, D. (2008) A DEVS-Based Framework for Simulation Optimization: Case Study of Link-11 Gateway Parameter Tuning. *IEEE Military Communications Conference*, San Diego, 16-19 November 2008, 1-7.

[14] Muzy, A. and Nutaro, J.J. (2005) Algorithms for Efficient Implementations of the DEVS & DSDEVS Abstract Simulators. *Proceedings of the* 1*st Open International Conference on Modeling & Simulation*, 273-279.

[15] Chow, A.C., Zeigler, B.P. and Kim, D.H. (1994) Abstract Simulator for the Parallel DEVS Formalism. *Fifth Annual Conference on AI, and Planning in High Autonomy Systems*, Gainesville, FL, USA, 7-9 December 1994, 157-163.

[16] Balakrishnan, V., Frey, P., Abu-Ghazaleh, N.B. and Wilsey, P.A. (1997) A Framework for Performance Analysis of Parallel Discrete Event Simulators. *Proceedings of the* 29*th Conference on Winter Simulation*, Atlanta, USA, December 1997, 429-436. https://doi.org/10.1145/268437.268520

[17] Himmelspach, J. and Uhrmacher, A.M. (2006) Sequential Processing of PDVES Models. *Proceedings of* 2*nd European Modelling and Simulation Symposium* (*EMSS* 2006), Barcelona, 4-6 October 2006, 239-244.

[18] Glinsky, E. and Wainer, G. (2006) New Parallel Simulation Techniques of DEVS and Cell-DEVS in CD++. 39*th Annual Simulation Symposium*, Huntsville, AL, USA, 2-6 April 2006, 244-251.

[19] Jafer, S. and Wainer, G. (2009) Flattened Conservative Parallel Simulator for DEVS and CELL-DEVS. *Proceedings of the* 2009 *International Conference on Computa-*

tional Science and Engineering, Volume 1, 443-448.
https://doi.org/10.1109/CSE.2009.52

[20] Jafer, S. and Wainer, G.A. (2011) A Performance Evaluation of the Conservative DEVS Protocol in Parallel Simulation of DEVS-Based Models. *Proceedings of the* 2011 *Symposium on Theory of Modeling and Simulation: DEVS Integrative M&S Symposium*, Boston, Massachusetts, April 2011, 103-110.

[21] Liu, Q. and Wainer, G.A. (2012) Multicore Acceleration of Discrete Event System Specification Systems. *Simulation*, **88**, 801-831.
https://doi.org/10.1177/0037549711412237

[22] Zacharewicz, G. and Hamri, M.E.-A. (2007) Flattening G-DEVS/HLA Structure for Distributed Simulation of Workflows. *Proceedings of AIS-CMS International Modeling and Simulation Multiconference*, Buenos Aires, 8-12 February 2007, 11-16.
http://hal.archives-ouvertes.fr/hal-00173659

[23] Al-Zoubi, K. and Wainer, G.A. (2010) Managing Simulation Workflow Patterns Using Dynamic Service-Oriented Compositions. *Winter Simulation Conference*, Baltimore, 5-8 December 2010, 765-777. https://doi.org/10.1109/WSC.2010.5679111

[24] Jafer, S. and Wainer, G. (2010) Global Lookahead Management (GLM) Protocol for Conservative DEVS Simulation. 2010 *IEEE/ACM* 14*th International Symposium on Distributed Simulation and Real Time Applications* (*DS-RT*), Fairfax, 17-20 October 2010, 141-148. https://doi.org/10.1109/DS-RT.2010.37

[25] Rao, D.M., Thondugulam, N.V., Radhakrishnan, R. and Wilsey, P.A. (1998) Unsynchronized Parallel Discrete Event Simulation. *Proceedings of the* 30*th Conference on Winter Simulation*, Washington DC, December 1998, 1563-1570.
https://dl.acm.org/doi/10.5555/293172.293536

[26] Franceschini, R., Bisgambiglia, P.-A. and Bisgambiglia, P. (2014) Decentralized Approach for Efficient Simulation of DEVS Models. *Proceedings in Advances in Production Management Systems. Innovative and Knowledge-Based Production Management in a Global-Local World*, Ajaccio, January 2014, 336-343.
https://link.springer.com/chapter/10.1007/978-3-662-44733-8_42
https://doi.org/10.1007/978-3-662-44733-8_42

[27] Bisgambiglia, P.-A., de Gentili, E., Bisgambiglia, P.A. and Santucci, J.-F. (2009) Fuzz-iDEVS: Towards a Fuzzy Toolbox for Discrete Event Systems. *Proceedings of the SIMUTools*'09, Rome, 3-5 March 2009, 10 p.
https://doi.org/10.4108/ICST.SIMUTOOLS2009.5691

[28] Barros, F.J. (1997) Modeling Formalisms for Dynamic Structure Systems. *ACM Transactions on Modeling and Computer Simulation*, **7**, 501-515.
https://doi.org/10.1145/268403.268423

[29] Bisgambiglia, P.-A., Franceschini, R., Chatel, F.-J., Rossi, J.-L. and Bisgambiglia, P. (2013) Discrete Event Formalism to Calculate Acceptable Safety Distance. *Proceedings of the* 2013 *Winter Simulation Conference*, Washington DC, 8-11 December 2013, 217-228. https://doi.org/10.1109/WSC.2013.6721421

[30] Garredu, S., Bisgambiglia, P.A., Vittori, E. and Santucci, J.F. (2009) A New Approach to Describe DEVS Models Using Both UML State Machine Diagrams and Fuzzy Logic. *Proceedings of HSC*, Huntsville, 27-29 October 2009, 156-162.

[31] Kofman, E., Giambiasi, N. and Junco, S. (2000) FDEVS: A General DEVS-Based Formalism for Fault Modeling and Simulation. *Proceedings of the European Simulation Symposium*, Ghent, Belgium, May 23-26 2000, 77-82.

[32] Zacharewicz, G., Hamri, M.E.-A., Frydman, C.S. and Giambiasi, N. (2010) A Generalized Discrete Event System (G-DEVS) Flattened Simulation Structure: Applica-

tion to High-Level Architecture (HLA) Compliant Simulation of Workflow. *Simulation*, **86**, 181-197. https://doi.org/10.1177/0037549709359357

[33] Wainer, G., Glinsky, E. and Gutierrez-Alcaraz, M. (2011) Studying Performance of DEVS Modeling and Simulation Environments Using the DEVStone Benchmark. *Simulation*, **87**, 555-580. https://doi.org/10.1177/0037549710395649

[34] Edijatno, De Oliveira Nascimento, N., Yang, X., Makhlouf, Z. and Michel, C. (1999) GR3J: A Daily Watershed Model with Three Free Parameters. *Hydrological Sciences Journal*, **44**, 263-277. https://doi.org/10.1080/02626669909492221

[35] Perrin, C., Michel, C. and Andréassian, V. (2003) Improvement of a Parsimonious Model for Streamflow Simulation. *Journal of Hydrology*, **279**, 275-289. https://doi.org/10.1016/S0022-1694(03)00225-7

[36] Andréassian, V., Perrin, C. and Michel, C. (2004) Impact of Imperfect Potential Evapotranspiration Knowledge on the Efficiency and Parameters of Watershed Models. *Journal of Hydrology*, **286**, 19-35. https://doi.org/10.1016/j.jhydrol.2003.09.030

[37] Bisgambiglia, P.-A., *et al.* (2017) DIMZAL: A Software Tool to Compute Acceptable Safety Distance. *Open Journal of Forestry*, **7**, 11-33. https://doi.org/10.4236/ojf.2017.71002

[38] Rothermel, R.C. (1972) A Mathematical Model for Predicting Fire Spread in Wildland Fuels. Res. Pap. INT-115, U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station, Ogden, 40 p.

[39] Franceschini, R., Bisgambiglia, P.-A., Bisgambiglia, P.A. and Hill, D.R.C. (2014) DEVS-Ruby: A Domain Specific Language for DEVS Modeling and Simulation (WIP). *Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative*, 1-6.

[40] Rossi, J.L., Simeoni, A., Moretti, B. and Leroy-Cancellieri, V. (2011) An Analytical Model Based on Radiative Heating for the Determination of Safety Distances for Wildland Fires. *Fire Safety Journal*, **46**, 520-527. https://doi.org/10.1016/j.firesaf.2011.07.007

[41] Foures, D., Franceschini, R., Bisgambiglia, P.-A. and Zeigler, B.P. (2018) multiP-DEVS: A Parallel Multicomponent System Specification Formalism. *Complexity*, **2018**, Article ID: 3751917. https://doi.org/10.1155/2018/3751917 https://www.hindawi.com/journals/complexity/2018/3751917/abs

[42] Franceschini, R., Bisgambiglia, P.-A., Bisgambiglia, P. and Hill, D.R.C. (2018) An Overview of the Quartz Modelling and Simulation Framework. *Proceedings of* 8*th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, Porto, 29-31 July 2018, 120-127. https://doi.org/10.5220/0006864201200127