Observations in DEVS framework

Gauthier Quesnel, Ronan Trépos, Patrick P. Chabrier, Jennifer Baudet, Raphaël Duboz

Observations in DEVS framework

Gauthier Quesnel[1], Ronan Trépos[1], Patrick Chabrier[1], Jennifer Baudet[1], Raphaël Duboz[2] and Éric Ramat[3]

[1]INRA, UR875 Biométrie et Intelligence Artificielle
F-31326 Castanet-Tolosan, France
gauthier.quesnel@toulouse.inra.fr

[2]ULCO, LISIC
50 rue Ferdinand Buisson BP 719
62228 Calais Cedex, France
ramat@lisic.univ-littoral.fr

[3]Cirad, Gestion Integrée des Risques
Campus de Baillarguet
34398 Montpellier cedex 5, France
raphael.duboz@cirad.fr

Keywords: DEVS, observation, methodology and simulation.

Abstract

The observation of a model is a necessary process in the context of modeling and simulation as it offers to modelers the results of their simulations. In this paper, we focus our works on the observation mechanism which is generally not explicit nor clearly specified. This is generally not an issue unless we want to use our model in experimental frames or to avoid the observation mechanism to interfere with the simulation. In this paper, we introduce an extension to the *Parallel Discrete Event System Specification* (PDEVS) formalism, to observe models in various ways, by event (at each state transition of a model), at the end of the simulation or by a time step. Thus, we define a formal specification of this extension and its abstract simulators algorithms. Finally, we present an implementation in the DEVS framework VLE.

1. INTRODUCTION

In the Modeling and Simulation (M&S) activity, observation of the behavior of models is a major task. Observation is the process that catches the state of a model during or at the end of the simulation. It allows the modeler to test, prove, validate or generate data from simulations by connecting simulation software to output streams like files, databases, unit test or visualisation softwares for instance. Observations contribute to the modeller's representation of its studied system. Nevertheless, the observation processes are generally not explicit nor specified in formalism or in simulation software. Thus, they are often model dependant and they simply consist in source code extension. However, on the major M&S software, observations are integrated in the simulation model like in Ptolemy II [7] or in Modelica [11].

This is generally not useful to specify formally the observation process. However, it becomes necessary when we want the observation to be modeled as a part of an experimental frame [13, 12, 1], where simulation results have to be captured following a precise experimental plan. For instance, in experimental frame, to calibrate a model or to make a sensitivity analysis, only a final observation of the system is necessary but in online optimisation by simulation or generally in dynamical experimental frames, we need the trajectories of variables of the models [5].

Our works take place in the M&S theory defined by B. P. Zeigler [13]. M&S theory tends to be as general as possible. It addresses major issues of computer sciences from artificial intelligence to model design and distributed simulations. M&S theory aims to develop a common framework, formal and operational, for the specification of dynamical systems. In this paper, we consider the *Discrete Event Specification System* (DEVS) formalism [13] from B. P. Zeigler works to develop a formal observation for the M&S theory. The DEVS formalism is well situated to clearly specify both system dynamics and experimental frames [13, 12]. Nevertheless, even if it is possible to specify observations using DEVS without any modification, it appears that the addition of a specific function dedicated to observation process can be very useful and computationally more efficient than a classic DEVS implementation (cf. figure 1). In this paper, we propose to extend the DEVS formalism with an observation process. These changes allow the modeler to observe its models in different ways.

2. METHOD

Before proposing the formalisation of the extension *observation*, we briefly describe, in the following section, the PDEVS formalism.

2.1. PDEVS

DEVS [13] is a well-known and accepted formalism for the specification of complex discrete or continuous systems abstracted as a network of concurrent, timed and interacting atomic and coupled models. PDEVS [3] extends the Classic DEVS [13] essentially by allowing bags of inputs to the external transition function. Bags can collect inputs which are built at the same date, and process their effects in future bags. This formalism offers a solution to manage simultaneous events that could not be easily managed with Classic DEVS. For a detailed description, we encourage the reader to read the chapter three in [13].

PDEVS defines an atomic model as a set of input and output ports and a set of state transition functions:

$$M = \langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$$

Where:

X is the set of input values
Y is the set of output values
S is the set of sequential states
$ta : S \rightarrow \mathbb{R}_0^+$ is the time advance function
$Q = \{(s,e) | s \in S, 0 \leq e \leq ta(s)\}$
  Q is the set of total states,
  e is the time elapsed since last transition
$\delta_{int} : S \rightarrow S$ is the internal transition function
$\delta_{ext} : Q \times X^b \rightarrow S$ is the external transition function
  X^b is a set of bags over elements in X
$\delta_{con} : S \times X^b \rightarrow S$ is the confluent transition function,
  subject to $\delta_{con}(s, \emptyset) = \delta_{int}(s)$
$\lambda : S \rightarrow Y$ is the output function

If no external event occurs, the system will stay in state s for $ta(s)$ time. When $e = ta(s)$, the system changes to the state δ_{int}. If an external event, of value x, occurs when the system is in the state (s,e), the system changes its state by calling $\delta_{ext}(s,e,x)$. If it occurs when $e = ta(s)$, the system changes its state by calling $\delta_{con}(s,x)$. The default confluent function δ_{con} definition is:

$$\delta_{con}(s,x) = \delta_{ext}(\delta_{int}(s), 0, x)$$

The modeler can prefer the opposite order:

$$\delta_{con}(s,x) = \delta_{int}(\delta_{ext}(s, ta(s), x))$$

Of course, this function can be completely defined by the modeler.

Every atomic model can be coupled with one or several other atomic models to build a coupled model. This operation can be repeated to form a hierarchy of coupled models. The set of atomic and coupled models and their connections is named the *structure of the model*. A coupled model is defined by:

$$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\} \rangle$$

Where X and Y are input and output ports, D the set of models and:

$\forall d \in D, M_d$ is a PDEVS model
$\forall d \in D \cup \{N\}, I_d$ is the influencer set of d :
  $I_d \subseteq D \cup \{N\}, d \notin I_d$
$\forall d \in D \cup \{N\}$,
  $\forall i \in I_d, Z_{i,d}$ is a function, the i-to-d output translation:
    $Z_{i,d} : X \rightarrow X_d$, if $i = N$
    $Z_{i,d} : Y_i \rightarrow Y$, if $d = N$
    $Z_{i,d} : Y_i \rightarrow X_d$, if $i \neq N$ and $d \neq N$

The influencer set of d is the set of models that interact with d and $Z_{i,d}$ specifies the types of relations between models i and d.

In previous section, we present succinctly the PDEVS formalism. In the following paragraphs and sections, we propose the formalization and abstract simulators of our observation extension.

2.2. Formal specification

The construction of a DEVS platform for modeling and simulation (like presented in section 3.) implies to observe models and their evolution during the simulation. Observation of DEVS models involves watching or capturing their states. These captures can be done when a change of state occurs (in DEVS terminology, in δ_{int}, δ_{ext} or δ_{con} transition functions) or at specific dates.

Observation of models is a necessary process in the context of the modeling and simulation. To develop a useful simulator software based on DEVS or which accepts DEVS models and theirs computation like OSA [10] or James II [6], for the modellers, the observation can be separated from the models behaviour. OSA or James II use classical software engineering paradigm to develop observation. Respectively "aspect oriented" techniques for OSA or the "design pattern observer" in James II. In this paper, we consider the development of the observation mechanism into the DEVS formalism itself. This work allows to clearly and formally defines when and how to observe models in a "all DEVS" vision of the simulation.

In DEVS, the commonly used solution (as presented in figure 1), is to connect models, both inputs and outputs, to an observation model. This model has in charge to send observation messages and to process the responses of the models. For instance, to build a discrete observation, the observation model relies on a constant ta function in order to send, at each time step, an event to observed models. Then, observed models output observation values that are sent to the observer.

This solution forces the modelers to merge the state graphs between observation and behavior of their models (See figure 2 for an explanation). In addition, by merging the state graphs, the modeler may make the results of its models dependent to the observation he makes. For example, if a model is observed asynchronously with its behaviour, the computation of the e or internal state variables, can give different results with or without observations. Precisely, this mismatch can be explained by the difficulty in representing reals in computer engineering [4] (see the IEEE Standard for Floating-Point Arithmetic, IEEE 754) .

Finally, in DEVS, a model can have several states at the same date (when at least one call to the $ta(s)$ function returns 0). These states are known as *transitory states*. It seems obvious that the observation of the transitory states makes no sense. However, this solution does not guarantee to provide

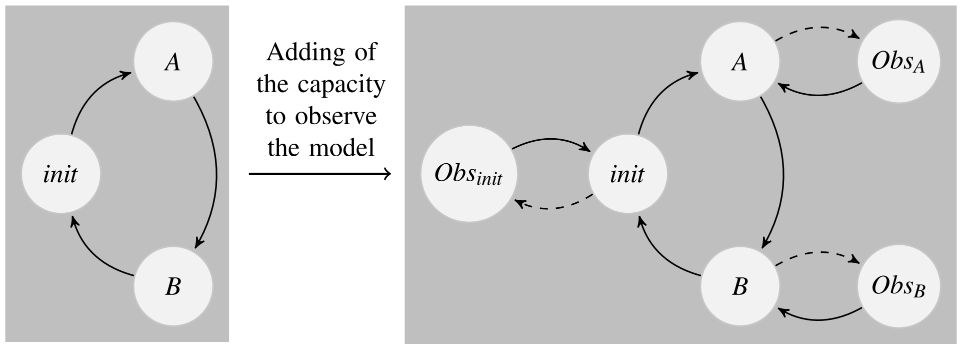


Figure 2. These pictures show state graphs of an atomic model. In left picture, we show the model without observation and in the right part, the same model with the capacity to be observed at any time (dotted lines represent external transitions). In the computation of the observation states Obs_{init}, Obs_A or Obs_B (in δ_{ext}, δ_{int}, δ_{conf} or ta functions), simulators can introduce floating-point errors when computing the time elapsed since last transition (e in DEVS terminology) for example.

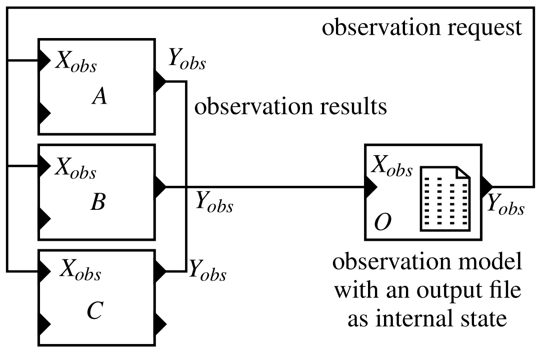


Figure 1. This picture shows the classic observation mechanism to observe models in the DEVS formalism. Three atomic models A, B and C are observed by an observation model O which sends event for request observation on output Y_{obs} and waits the results on input port X_{obs} in order to store it in output files, databases etc.

only observations of real model states. For example, if the observation is required at a specific time of simulation, the state observed can be one of these transitory states.

In this section, we propose a formal and operational representation of observations in the PDEVS formalism. This extension is based on a function of observation in atomic models and a second graph of connections in coupled models. This extension to the PDEVS formalism integrates necessary characteristics:

- first is to ensure that this extension does not disturb the simulation i.e. we need to conserve the same result with or without the observation.
- second is to provide several types of observation. We propose, in this paper, two modes: *time step* and *finish* which define respectively: observations when a model reaches a time and observations at the end of the simulation. (See figure 3).
- finally, in DEVS, a model may change several times statements to the same date (when its ta returns 0 or an external events disturb it). In this case, we observe the last state of the model.

Thus, to develop this extension we need to add function to the atomic models, to add new set in coupled model and add specific observation models. We develop these changes in the next section.

2.2.1. PDEVS Atomic model

To develop this extension in the PDEVS formalism, we extend the PDEVS atomic model with new sets of input and output ports ($X_{obs?}$ and Y_{obs}). The first one is used for observation request, the second one is used for routing values returned by modeler. We attach to this port a new output function called λ_{obs}. The new PDEVS atomic model is defined such as:

$$M = \langle X, Y, S, X_{obs?}, Y_{obs}, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, \lambda_{obs}, ta \rangle$$

Where:

$X_{obs?}$ are the ports used to catch observation requests
Y_{obs} are the ports used to send observation values
$\lambda_{obs} : X_{obs?} \times s \times t_{obs} \rightarrow Y_{obs}$
$s \in S$,
t_{obs} current time of the simulation.

This definition allows us to represent both observed models and classical PDEVS models as defined in section 2.1.. Indeed the model $\langle X, Y, S, X_{obs?}, Y_{obs}, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, \lambda_{obs}, ta \rangle$ with no observations[1] is equivalent to the classical PDEVS

[1] $X_{obs?} = \emptyset$, $Y_{obs} = \emptyset$ and λ_{obs} is a null function.

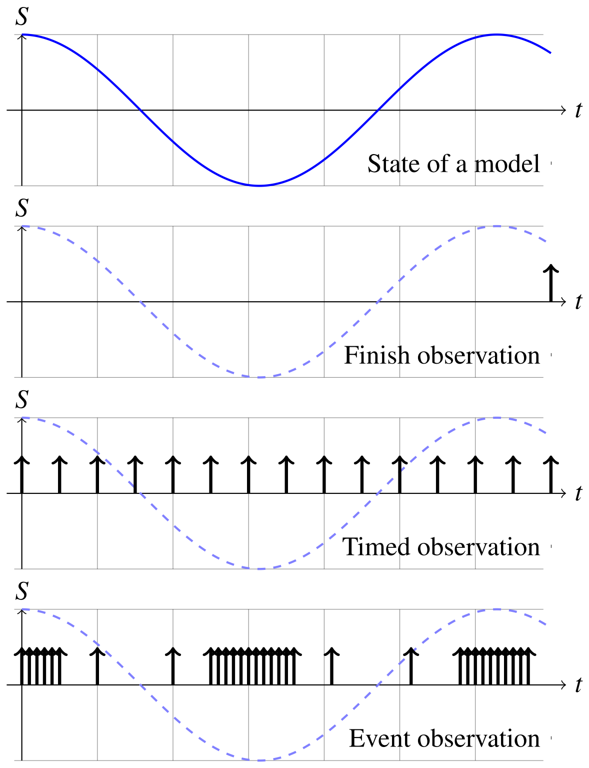


Figure 3. This figure shows three modes for observing the state of a system (in the top chart). In the next charts, we use an arrow to show when an observation occur. Thus, in the second chart, we show a finish observation at the end of the simulation, in the third chart, we show a time step observation with a duration of $\Delta_t = 0.5$ between two observations and finally, we show an event view, when the value of the observed variable crosses thresholds.

atomic model $\langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta\rangle$. In fact, we should distinguish observed models from observer models.

2.2.2. Observer models

In this paper, we suggest two types of observers. O_{timed}, and O_{finish} to respectively, send observation following a given time-step or at the end of the simulation.

- The discrete time observer model, O_{timed}, needs to send output events on its output port Y_{obs} every Δ_t unit time. O_{timed} reads observation events from its port X_{obs} and stores data transported by event. This model is :

$$O_{timed} = \langle X_{obs}, Y_{obs?}, S, X_{obs?}, Y_{obs}, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, \lambda_{obs}, ta\rangle$$

Where:

X_{obs} is the set of observation values,
$Y_{obs?}$ is the set of observation requests,
$S : \{IDLE, SENT\} \times Data$, with:
$IDLE, SENT$: automata finite states,
$Data$: output stream states (files, database, etc.).
$X_{obs?} = \emptyset$
$Y_{obs} = \emptyset$
$\forall d \in Data : \delta_{int}((IDLE, d)) = (SENT, d)$
$\forall d \in Data : \delta_{int}((SENT, d)) = (IDLE, d)$
$\forall s \in \{IDLE, SENT\} : \delta_{ext}((s, d), e, X_{obs}) = (s, d')$
where d' is the state of the output stream once X_{obs} has been stored,
$\forall d \in Data : ta((IDLE, d)) = \Delta_t$ (the time step)
$\forall d \in Data : ta((SENT, d)) = 0$
$\forall s \in S : \lambda(s) = build\_request(s)$
λ_{obs} is a null function

- The finish observer model, O_{finish}, produces observation events at the end of the simulation on it Y_{obs} port and wait observation events from its port X_{obs}. O_{finish} is the same model as O_{timed} but, its ta function returns the end of the simulation to build observation event, or $+\infty$ after.

$$O_{timed} = \langle X_{obs}, Y_{obs?}, S, X_{obs?}, Y_{obs}, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, \lambda_{obs}, ta\rangle$$

Where:

X_{obs} is the set of observation values,
$Y_{obs?}$ is the set of observation requests,
$S : \{WAIT, END\} \times Data$, with:
$WAIT, END$: automata finite states,
$Data$: output stream states (files, database, etc.).
$\forall d \in Data : \delta_{int}((WAIT, d)) = (END, d)$
$\forall d \in Data : \delta_{int}((END, d)) = (END, d)$
$\forall s \in \{WAIT, END\} : \delta_{ext}((s, d), e, X_{obs}) = (s, d')$
where d' is the state of the output stream once X_{obs} has been stored,
$X_{obs?} = \emptyset$
$Y_{obs} = \emptyset$
$\forall d \in Data : ta((WAIT, d)) = end$
where end is the duration of simulation
$\forall d \in Data : ta((END, d)) = +\infty$
$\forall s \in S : \lambda(s) = build\_request(s)$
λ_{obs} is a null function

The function $build\_request$ is a user defined function that identifies in $Y_{obs?}$ the observation request that should be sent to the observed models.

2.2.3. Coupled model

As seen earlier in the introduction and in the section 2.1., the I_d variable and the function $Z_{i,d}$ define the graph of connections of the models by calculating the influencees models. Our observation extension of the PDEVS formalism uses the same principle. It proposes two subsets I_o and I_r which, respectively identify the influencees of atomic models (responses of the observations) and the influencees of observer models (models sending the request of observation).

We extend the PDEVS model coupled in order to introduce a second connection network (see the figure 4). This second network is dedicated to the observer models and to the classical PDEVS atomic models to observe through their ports $X_{obs?}$ and Y_{obs}. The new PDEVS coupled model is defined as:

$$\begin{aligned} N = \langle & X, Y, D, O, R, X_{obs?}, X_{obs}, Y_{obs?}, Y_{obs}, \\ & \{M_d\}, \{I_d\}, \{Z_{i,d}\}, \\ & \{M_o\}, \{O_d\}, \{Z_{o,d}\}, \{M_r\}, \{R_d\}, \{Z_{r,d}\} \rangle \end{aligned}$$

Where X_{obs}, Y_{obs} and $X_{obs?}$, $Y_{obs?}$ are input and output ports to route observation and request events. O is the set of names of observed models $O \subseteq D$, $\{M_o\}$ is the set of observed models $\{M_o\} \subseteq \{M_d\}$, R is the set of names of observer models $R \subseteq D$ and $\{M_r\}$ is the set of observer models $\{M_r\} \subseteq \{M_d\}$.

$$\begin{aligned} & \forall r \in R \cup \{N\}, \\ & \quad I_r \text{ is the influencer set of observed models of } r: \\ & \quad I_r \subseteq O \cup \{N\}, r \notin I_r \\ & \forall o \in O \cup \{N\}, \\ & \quad I_o \text{ is the influencer set of observed models of } o: \\ & \quad I_o \subseteq R \cup \{N\}, o \notin I_o \end{aligned}$$

The observation network is however very constrained. Thus, the functions $Z_{o,d}$ and $Z_{r,d}$ are very different and provide additional constraints from the classical $Z_{i,d}$ function. These additional constraints ensure that a DEVS atomic model cannot send an observation event to another DEVS atomic model, and an observation model cannot send request observation to another observation model. The figure 4 shows an example of connections between observed and observer models.

$$\begin{aligned} & \forall r \in R \cup \{N\}, \\ & \quad \forall d \in I_r, \exists \text{ an output translation function } Z_{r,d}: \\ & \qquad Z_{r,d} : X_{obs?} \rightarrow X_{obs?_d}, \text{ if } r = N \\ & \qquad Z_{r,d} : Y_{obs?_r} \rightarrow Y_{obs?}, \text{ if } d = N \\ & \qquad Z_{r,d} : Y_{obs?_r} \rightarrow X_{obs?_d}, \text{ if } r \neq N \text{ and } d \neq N \\ & \forall o \in O \cup \{N\}, \\ & \quad \forall d \in I_o, \exists \text{ an output translation function } Z_{o,d}: \\ & \qquad Z_{o,d} : X_{obs} \rightarrow X_{obs_d}, \text{ if } o = N \\ & \qquad Z_{o,d} : Y_{obs_o} \rightarrow Y_{obs}, \text{ if } d = N \\ & \qquad Z_{o,d} : Y_{obs_o} \rightarrow X_{obs_d}, \text{ if } o \neq N \text{ and } d \neq N \end{aligned}$$

Closure under coupling

As we do not change the internal S of coupled or atomic models nor all sets of the PDEVS definition (IMM, $INF(s)$, $CONF(s)$, $EXT(s)$ and $UN(s)$ of the coupled model), the PDEVS functions have the same content (See chapter 7 in [13] to the complete formalization), and the closure under coupling property of the extension observation for parallel DEVS is still verified.

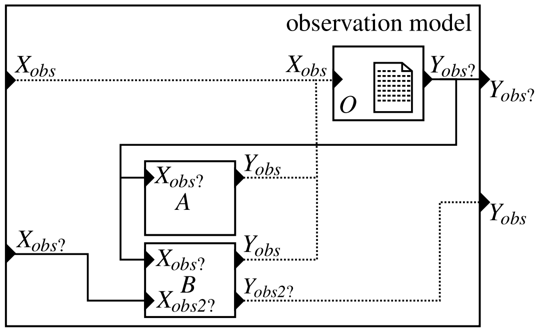


Figure 4. In this picture, we illustrate the second network in the coupled model to deal with observations graph. Plain connections are observation requests from output port $Y_{obs?}$ of observation models, to atomic model input port $X_{obs?}$ or to coupled model's output port $Y_{obs?}$, or from coupled model input port $X_{obs?}$ to atomic model $X_{obs?}$. Dashed connections constitute response of observation request from extended PDEVS atomic model (on output port Y_{obs}) to observation model X_{obs} or coupled model output port Y_{obs}. The atomic model B is observed by an external observation model on its port X_{obs2}. It sends observations to its output port Y_{obs2}. For this figure and for a better understanding, we distinguish observation ports for the model B.

2.3. Algorithms

2.3.1. Root coordinator

The root-coordinator implements the overall simulation loop. It sends messages to its direct subordinate (simulator or coordinator). The root-coordinator first sends an initialize message (*i-message*), and loop on internal transition (**-message*) from its child to perform the simulation cycles until some termination conditions.

```
Parallel−Devs−Root−Coordinator
variables:
  t // current simulation time
  child // direct subordinate devs−simulator or devs−coordinator

  t = $t_0$
  send initialization message (i,t) to child
  t = tn of its child
  loop
    send (∗,t) message to child
    t = tn of its child
  until end of simulation
end Parallel−Devs−Root−Coordinator
```

As describe in previous algorithm, the root coordinator is not modified in this extension.

2.3.2. Coordinator

In the coordinator algorithm, we add an additional variable *tb* which indicates the last date of the simulation. This variable *tb* detects the last PDEVS bags to ensure to observe the latest state of a model. We add a new bag, called *mailobs* to store all observation events to atomic models at a specific date. When *tb* detect a change, the *mailobs* is flushed into the observation network using the functions $Z_{r,d}$ and $Z_{o,d}$.

```
1  Parallel−Devs−Coordinator
2  variables:
3    $DEVN = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})$
4    parent // parent coordinator
5    tl // time of last event
6    tn // time of next event
7    eventlist // list of element (d, $tn_d$) sorted by $tn_d$
8    mail // output mail bag
9    mailobs // observation mail bag
10   $O_{event}$ // event−list of event observation model
11   $y_{parent}$ // output message bag to parent
12   $y_d$ // set of output message bags for each child d
13
14   when receive xobs−message $(x_{obs}, t)$
15     if not $(tl \leq t \leq tn)$ then
16       error: bad synchronisation
17     receivers = $\{o | o \in children, o \in I_r\}$
18     for each $o \in$ receivers
19       send x−message $(Z_{r,d}(x)$ with input value $Z_{r,d}$ to $o$
20
21   when receive xobs?−message $(x_{obs?}, t)$
22     if not $(tl \leq t \leq tn)$ then
23       error: bad synchronisation
24     receivers = $\{r | r \in children, r \in I_o\}$
25     for each $r \in$ receivers
26       add $r, x_{obs?}$ to mailobs
27
28   when receive yobs−message $(y_{obs}, t)$ from $o$
29     for each child $d \in Z_{o,d}$
30       send xobs−message to d
31     for each $d \in Z_{o,d}$ and $d = N$
32       send yobs−message to parent
33
34   when receive yobs?−message $(y_{obs?}, t)$ from $r$
35     for each child $d \in Z_{r,d}$
36       send yobs−message to d
37     for each $d \in Z_{r,d}$ and $d = N$
38       send yobs?−message to parent
39
40   // finally, we append these algorithms in the following message
41
42   when receive i−message (i,t) at time t
43     [...] // The same algorithms than PDEVS
44     tb = tl
45
46   when receive ∗−message, x−message or y−message
47     if tb != t then
48       for each $r, x_{obs?} \in$ mailobs
49         send xobs?−message $(Z_{r,tb}(x)$ with input value $Z_{r,d}$ to $r$
50       $mailobs = \emptyset$
51       $tb = t$
52     [...] // The same algorithms than PDEVS
53
54 end Parallel−Devs−Coordinator
```

- from l. 1 to l. 38: routes the message or stores in *mailobs*.
- from l. 40 to the end: to update the *tb* variable and to send *xobs?-message* to the atomic models.

2.3.3. Simulator

Lastly, this last section on the abstract simulators develops algorithm for the simulator of the atomic model. As presented previously, the management of the observations events is very simple for the modeler since only one function is called λ_{obs}. This function is called only for classical atomic models when they receive an $X_{obs?}$ event. When observation models receives X_{obs} event, they use the classical way when receiving input message (See *x-message* in PDEVS abstract simulators).

```
1  Parallel−Devs−Simulator
2  variables:
3    parent // parent coordinator
4    tl // time of last event
5    tn // time of next event
6    DEVS // associated model with total state (s,e)
7    $y_{obs}$ // output observation bag
8
9    [...] // The same algorithms than PDEVS simulator see chapter 11.
10
11   when receive xobs?−message $(x_{obs?}, t)$ at $t$ with input $x_{obs?}$
12     $y_{obs} = \lambda_{obs}(x_{obs?}, t)$
13     send yobs−message $(y_{obs}, t)$ to parent coordinator
14
15 end Parallel−Devs−Simulator
```

- l. 12: when receives an *xobs?-message* from the parent, simulator computes the observation in the λ_{obs} function and sends the data (wrapped into an *yobs-message*) to the parent coordinator.

To validate this work, these abstract simulators are implemented in VLE, a platform of modeling and simulation VLE.

3. RESULT

Our works have started with multi-disciplinary issues emerging from the interaction between computer scientists and biologists. Considering these works, we think that the integration of heterogeneous models and the respect of the M&S cycle are the key issues to provide a complete and reliable software environment for natural complex systems modelling. So, VLE (*Virtual Laboratory Environment*) has evolved toward a complete multi-modelling and simulation environment [8] and is now a generic environment for M&S, in Environmental Sciences, in Industry or Medicine. It is used in many projects from two major French research institutes INRA and Cirad like the RECORD project [2]. RECORD is a platform designed for developing models of cropping systems, including crops, soils, pests, pathogens, and farmers, at different spatial and temporal scales. Scientists will use the RECORD platform to develop new models as modular components, to re-use and combine them in order to represent cropping systems and to share them with the community.

Technically, VLE is a set of softwares and libraries which supports multi-modeling and simulation by implementing the PDEVS abstract simulators and a DEVS bus. VLE is oriented toward the integration of heterogeneous formalisms like integration of ordinary differential equations (DESS, QSS 1 and 2), petri nets, finite state automata (moore, mealy, UML statechart), cellular automata (Cell-DEVS and Cell-QSS) and agent. Furthermore, VLE is able to integrate specific models developed in most popular programming languages into one single multi-model. Finally, VLE uses an open-source license and all the source code are available on its website including this observation extension (see the figure 5 for screenshots of VLE).

3.1. Implementation

In our implementation of the PDEVS abstract simulator, we define an atomic model as a classical C++ class. This class must be inherited to override the observation function and to benefit to the observation extension developed in the previous section.

```
1 class Dynamics {
2 public:
3 Time timeAdvance() const;
4 void internalTransition(const Time& time);
5 void externalTransition(const Time& time, const ExternalEventList& lst);
6 void output(const Time& time, ExternalEventList& out) const;
7 void confluentTransition(const Time& time, const ExternalEventList& lst);
8 Value* observation(const ObservationEvent& event) const;
9 };
```

This simplified class defines:

- l. 4-7: the classic PDEVS functions: ta, δ_{int}, δ_{ext}, λ and δ_{con}.
- l. 8: the observation method is called when an *observation event* arrived on a input observation port X_{obs}. λ_{obs} is also a *constant* function to prevent user to modify the state of its model. This function returns a Value (simple type as integer, real, boolean, string, and complex type as set, dictionary, matrix etc.).

3.2. Experimental frames in VLE

The study of the models is very important in the modeling and simulation cycle. It is generally carried out using experimental frames. Experimental frames as sensitivity analysis, replicas generation, etc. are used to study the possibilities of models. One important motivation in formalising the observation process in PDEVS is to develop generic experimental frames, which ones rely on intensive observation of models. In the VLE environment, we propose tools for managing the experimental frames. These tools provide methods to:

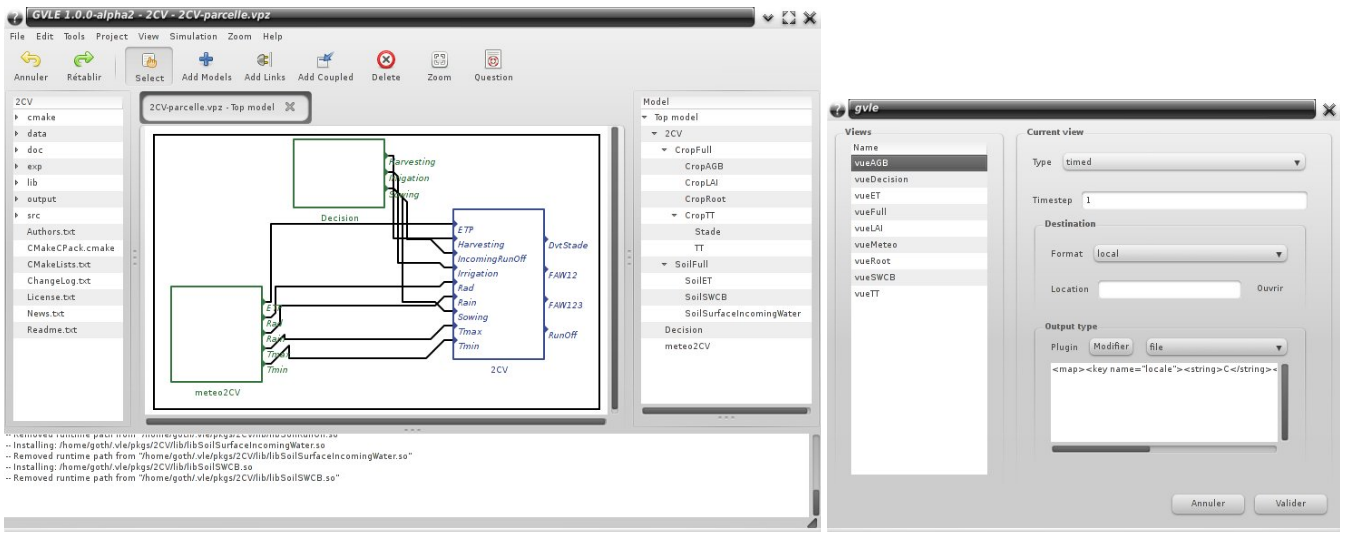
- Assign parameters or experimental conditions to atomic models.
- Define observation of atomic models, frequency, type, output.
- Build instances of the experimental design by performing combinations between different experimental conditions and applying a number of replicas with particular seed.
- Execute these instances of the experimental design on the grid computation composed of workstations and/or clusters (distributed and parallelized).

The VLE environment is based on a set of libraries called VFL (*VLE Foundation Libraries*). The development of new programs, based on these libraries, is easily achievable. Thus, in order to collaborate with users of statistical tools, we provide an interface to the program R [9]: a tool and language for statistical computing. This package, called RVLE, has the same capabilities as the VFL and provided an easy-to-use tool to exploit and to explore model output based on VLE simulations. In this context, the observation extension is fundamental. The link between the simulator and the scripting language used a piece of software to transform data of observations to R data frames. Following the same idea, we propose, *pyvle* and *jvle* respectively for Python and Java interpreters.

4. CONCLUSION AND PERSPECTIVES

In this paper, in section 2., we extend the PDEVS formalism to introduce the observation of models in the formalism. This development was motivated by the separation between the dynamics of the system and its observation as a fundamental issue. This addition to the formalism is closed under coupling and allows to build observations in a hierarchical and modular manner. The abstract simulators necessary to run were also formally described. And a implementation respectful to the proposed add-on has been provided.

This DEVS extension is crucial for us to develop experimental design and to abstract observation from models' dynamic. However, a type of observation is missing in these works. It concerns the event observation of atomic models. In PDEVS terminology, after each change in δ_{int}, δ_{ext} or δ_{con} transition functions. We work to formalize this new type of observation. It will be done in a future paper.



```
-- Installing: /home/goth/.vle/pkgs/2CV/lib/libSoilSurfaceIncomingWater.so
-- Removed runtime path from "/home/goth/.vle/pkgs/2CV/lib/libSoilSurfaceIncomingWater.so"
-- Installing: /home/goth/.vle/pkgs/2CV/lib/libSoilSWCB.so
-- Removed runtime path from "/home/goth/.vle/pkgs/2CV/lib/libSoilSWCB.so"
```

Type: timed
Timestep: 1
Format: local
Location:
Plugin: file

```
<map><key name="locale"><string>C</string><
```

Figure 5. These pictures show the GVLE modeling tool. In the left picture, GVLE is used to build hierarchy of models and connections, to attach behaviours, experimental conditions and observations to atomic models. In the right picture is used to parametrize the observer models.

Finally, from the user point of view of the M&S framework the delivery of a reliable service of observation that can be used either in simple case or in complex experimental frame is a true advantage when the need is to study simulations of heterogeneous coupled models.

REFERENCES

[1] B. Bonté, R. Duboz, G. Quesnel, and J.P. Müller. Recursive simulation and experimental frame for multiscale simulation. In *Summer Computer Simulation Conference*, pages 164–172, Istanbul, Turkey, July 2009.

[2] P. Chabrier, F. Garcia, R. Martin-Clouaire, G. Quesnel, and H. Raynal. Toward a simulation modeling platform for studying cropping systems management: the record project. In *International Congress on Modelling and Simulation, International Society for Computer Simulation*, pages 10 –13, Christchurch. New Zealand., 2007.

[3] A.C.H. Chow and B.P. Zeigler. Parallel DEVS: a parallel, hierarchical, modular, modeling formalism. In *Proceedings of the 26th conference on Winter simulation*, pages 716–722, Orlando, Florida, United States, 1994.

[4] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23:5–48, March 1991.

[5] A. Gosavi. *Simulation-Based Optimisation – Parametric Optimization Techniques and Reinforcement Learning*. Kluwer Academic Publishers, 2003.

[6] J. Himmelspach and A. M. Uhrmacher. The JAMES II Framework for Modeling and Simulation. In *International Workshop on High Performance Computational Systems Biology*, pages 101–102, 2009.

[7] Edward A. Lee. Overview of the Ptolemy Project, july 2003. Technical Memorandum No. UCB/ERL M03/25.

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

[9] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2006. ISBN 3-900051-07-0.

[10] J. Ribault, O Dalle, D. Conan, and S. Leriche. OSIF: a framework to instrument, validate, and analyze simulations. In *SIMUTools '10 Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, 2010.

[11] Michael M. Tiller. *Introduction to Physical Modeling with Modelica*. Kluwer Academic, 2001.

[12] M. K. Traoré and B. P. Zeigler. Experimental Frames Methodology. In *NSF Workshop on Modeling and Simulation for Design of Large Software-Intensive Systems: Challenges and New Research Directions. DLS03*, Tucson, AZ, USA, December 2003.

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