Observations of Discrete Event Models

Gauthier Quesnel[1], Ronan Trépos[1] and Éric Ramat[2]
[1]*INRA, UR875 Biométrie et Intelligence Artificielle, F-31326 Castanet-Tolosan, France*
[2]*ULCO, LISIC, 50 Rue Ferdinand Buisson BP 719, 62228 Calais Cedex, France*

Keywords: Simulation, Discrete Event Systems, DEVS Formalism, Observation, Methodology.

Abstract: The observation of a simulation is an important task of the modeling and simulation activity. However, this task is rarely explained in the underlying formalism or simulator. Observation consists to capture the state of the model during the simulation. Observation helps understand the behavior of the studied model and allows improving, analyzing or debugging it. In this paper, we focus on appending an observation mechanism in the *Parallel Discrete Event System Specification* (PDEVS) formalism with guarantee of the reproducible simulation with or without observation mechanism. This extension to PDEVS allows us to observe models at the end of the simulation or according to 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 an important aspect. An observation captures the state or the evolution of the state 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 application to output streams like files, databases, unit test or visualization software. Observations contribute to the modelers representation of its studied system.

In environmental and agronomic modeling domain, we need to couple heterogeneous models. These models can be continuous or discrete. In addition, these models co-exist in different temporal and spatial scales. Thus, to solve this problem, we use the *Discrete Event Specification System* (DEVS) formalism (Zeigler et al., 2000). DEVS can be seen as a common denominator for multi-formalism hybrid systems modeling (Vangheluwe, 2000). DEVS takes place in the M&S theory defined by B. P. Zeigler. 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.

Modeling the observation process is dependent on the underlying formalism. However, for many formalisms, the observation does not exist and the outputs of models are used directly. For example, the numerical integration method Euler used for solving ordinary differential equations does not allow catching variable between two time-steps. The observation process is generally not explicit nor specified in the formalism. Specify the observation process is generally not useful. However, it becomes necessary when we want to observe multi-models with the same observation mechanism.

However, the DEVS formalism does not propose observation of simulation models. Several solutions exist without changing the behavior of DEVS. For example, in the first picture of the figure 1, outputs of the models A and B are connected to an observation model O. O has a reactive behavior. PowerDEVS (Bergero and Kofman, 2010) chooses this solution. It is simple and has great flexibility. However, it can be very expensive. Indeed, in the case of experimental frames (to calibrate or estimate parameters or to optimize parameters on criteria), only the final values of the observations are necessary. All the values received are useless. The interpretation of these values can be costly. In the second picture of the figure 1, an observation model O generates events to observed models A and B. These models send their outputs to the observation model. This solution is computationally less expensive, but requires mixing observation and normal behavior of the model. This mechanism can induce a difficulty in reproduc-

Quesnel G., Trépos R. and Ramat É..
Observations of Discrete Event Models.
DOI: 10.5220/0004054300320041
In *Proceedings of the 2nd International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH-2012), pages 32-41
ISBN: 978-989-8565-20-4

ing simulations and can increase the complexity of model.

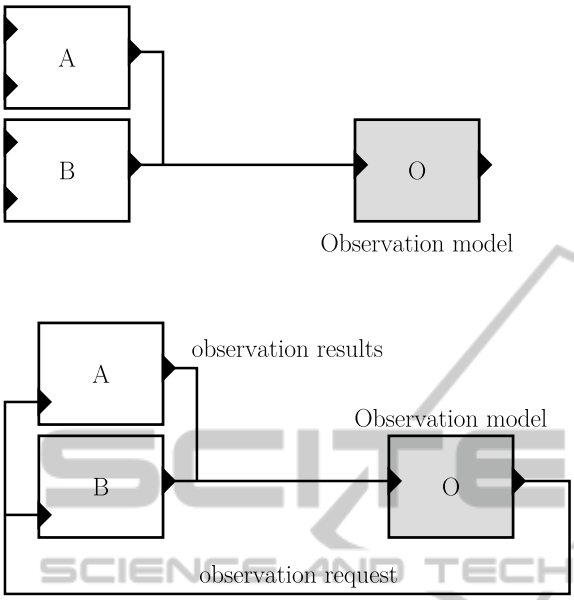

Figure 1: Classical reactive and proactive observation mechanism in DEVS.

We chose in this work to modify the DEVS formalism by adding an extension. This addition does not change the behavior of DEVS. It adds a layer of observation to DEVS. The DEVS trajectories of the simulations are the same with or without this extension. However, some alternative exist to add an observation mechanism. Thus, rather than developing the observation process in the formalism, researchers developed this process in the software part of their simulators. It is the choice made by JAMES 2 (Himmelspach and Uhrmacher, 2009) and OSA (Ribault et al., 2010). JAMES 2 uses the design pattern observation; OSA uses AOP (*Aspect-oriented programming*. The design pattern observation links for each model a list of observation component. This list has a function called `notify`. The model calls this function when it wants to send observation value. The observation component has a reactive behavior. This type of behavior can increase the calculation to perform the treatment of observation values from multi-models (e.g. Models that use formalism with different time advance function). Of course, through a combination of the visitor and observer design patterns, we can make active the behavior of the observation component.

In this paper, we propose to extend the PDEVS formalism with an observation mechanism. This extension to DEVS provides a generic way to observe heterogeneous models and it guarantees the same simulations when this mechanism is activated and when it is not.

This paper is divided into three parts. In the first part, we explain the PDEVS formalism. This formalism is the context of our work. Then, we develop our contribution in two parts. (1) We develop the formal specification of the observation extension. This specification guarantees to conserve the behavior of PDEVS. (2) We present the abstract simulators of the observation extension. Developers of PDEVS simulators software can use these abstract simulators to introduce the extension observation in their simulators. The last sections show an implementation of this extension in the software VLE and an example of use. Finally, we conclude this paper with conclusion and perspectives.

2 METHOD

DEVS (Zeigler et al., 2000) 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 models. DEVS provides a hierarchical and modular approach to modeling and simulation. The formalism supports two types of models: atomic and coupled models. Atomic models interact thought a well-defined set of input and output values enabling a software component based orientation to model representation. Atomic models can be combined to form coupled models, providing the constructs to the representation of complex models.

2.1 PDEVS Formalism

PDEVS (Chow and Zeigler, 1994) extends the Classic DEVS essentially by allowing bags of inputs to the external transition function. Bags can collect inputs that 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 section 3.4.2 in chapter 3 and the section 11.4 in chapter 11 of Zeigler's book (Zeigler et al., 2000).

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

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

With:

X is the set of input values

Y is the set of output values

S is the set of sequential states

$ta : S \to \mathbb{R}_0^+$ is the time advance function

$\delta_{int} : S \to S$ is the internal transition function

$\delta_{ext} : Q \times X^b \to S$ is the external transition 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

X^b is a set of bags over elements in X

$\delta_{con} : S \times X^b \to S$ is the confluent transition function, subject to $\delta_{con}(s,\emptyset) = \delta_{int}(s)$

$\lambda : S \to Y$ 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:

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

Indeed, the modeler can define its own function.

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. 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:

$Z_{i,d} : X \to X_d$, if $i = N$

$Z_{i,d} : Y_i \to Y$, if $d = N$

$Z_{i,d} : Y_i \to X_d$, 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.

2.2 Extension Observation: Formal Specification

The development of simulation software based on DEVS formalism 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.

In DEVS, the solution generally used is to connect models, both inputs and outputs, to an observation model (see figure 2). This model has in charge to send observation messages and to process the responses of the models. For example, 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. Observed models compute observations values and send them to the observer.

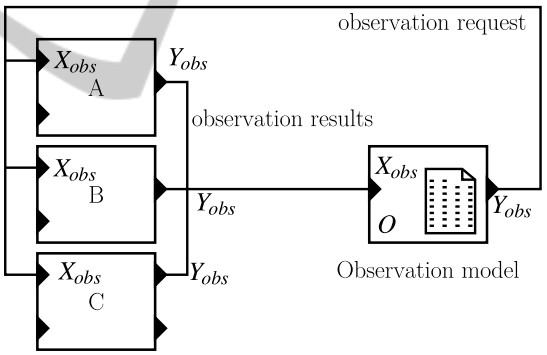

Figure 2: This picture shows the classic observation mechanism to observe models in the DEVS formalism. Three atomic models A, B and C are observed by an observation model O which sends event for request observation on output Y_{obs} and waits the results on input port X_{obs} in order to store it in output files or databases.

This solution forces the modelers to merge the state graphs between observation and behavior of their models (See figure 3 for an explanation).

In addition, by merging the state graphs, the modeler might make the results of its models dependent to his observation. If model is observed asynchronously with its behavior, the computation of the elapsed time e when an external event disturbs an atomic model may introduce a floating-point error. Indeed, coordinator needs to call the external transition of the simulator with s,e,x parameters where $e = t - tl$ with t the

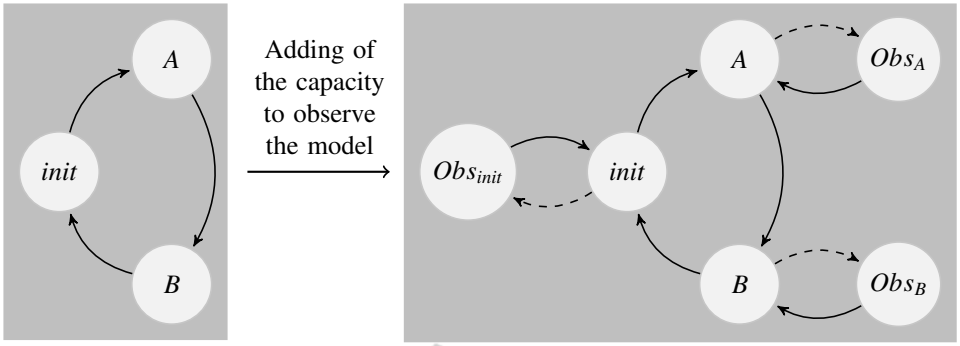

Figure 3: This picture show state graphs of an atomic model. In left picture, we show the model without observation and in the right part, the same model with the capacity to be observed at any time (dotted lines represent external transitions). In the computation of the observation states Obs_{init}, 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.

current time of the simulation (time of the perturbation) and *tl* the time of last event. If an internal state variable in *S* is based on *e* to compute its value in the δ_{ext} transition, its internal state variables can give different results with or without observations. Precisely, this mismatch can be explained by the difficulty representing real number in computer engineering (Goldberg, 1991) (see the IEEE Standard for Floating-Point Arithmetic, IEEE 754).

If all models use integer as representation of the time (i.e. all time advance functions return duration in $\mathbb{N}$), the floating-point problem would not exist. However, in the domain of Environmental Modeling, the models use continuous and discrete event formalisms such as the QSS2 formalism (Kofman, 2002) which solves in $\mathbb{R}$ a second order polynomial function in order to compute the duration in a state. They can hardly rely on a representation of the time in $\mathbb{N}$.

The solution proposed does not change the value of *e* in the case of observation.

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 called *transitory states*. It seems obvious that the observation of the transitory states makes no sense. However, this solution does not guarantee to provide 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* that define respectively: observations when a model reaches a time and observations at the end of the simulation (see figure 4).
- Third, the extension should not take its values directly from the state of the current model but from a function (see figure 4).
- Finally, in DEVS, state variables may change several times at the same date (when its *ta* returns 0 or when an external event 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.

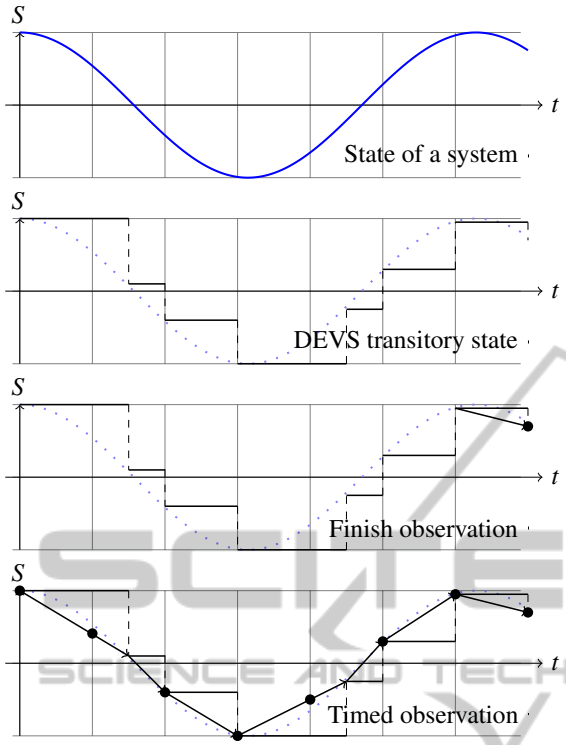

Figure 4: The first figure shows the evolution of the state S of a system over time t. The second figure shows the model of this system in DEVS. This figure shows the trajectory of the state of the DEVS model. The last two figures show the extension observation described in this paper in *time-step* ($\Delta_t = 1$) or in *finish* mode. This extension allows the modeler to provide a function (`observation`) that computes an observation value. Dots are observation values. These values are computed by interpolation of the current DEVS state of s and e, the elapsed time since the last transition.

$$\lambda_{obs} : X_{obs?} \times s \times t_{obs} \rightarrow Y_{obs}$$
$s \in S$,
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 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.

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

- 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.
$X_{obs?} = \emptyset$
$Y_{obs} = \emptyset$
$\forall d \in Data : \delta_{int}((IDLE, d)) = (SENT, d)$
$\forall d \in Data : \delta_{int}((SENT, d)) = (IDLE, d)$
$\forall s \in \{IDLE, SENT\}$:
$\delta_{ext}((s,d), e, X_{obs}) = (s, d')$
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)
$\forall d \in Data : ta((SENT, d)) = 0$
$\forall s \in S : \lambda(s) = observation(s, t, x)$
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.

$\forall d \in Data : \delta_{int}((WAIT, d)) = (END, d)$
$\forall d \in Data : \delta_{int}((END, d)) = (END, d)$
$\forall s \in \{WAIT, END\}$:
$\delta_{ext}((s,d), e, X_{obs}) = (s, d')$

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
$\forall d \in Data : ta((END, d)) = +\infty$
$\forall s \in S : \lambda(s) = observation(s, t, x)$
λ_{obs} is a null function

The function *observation* 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 influenced models. Our observation extension of the PDEVS formalism uses the same principle. It proposes two subsets I_o and I_r that, respectively identify the influences of atomic models (responses of the observations) and the influences 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 5). 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:

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

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 $\{M_r\} \subseteq \{M_d\}$.

$\forall r \in R \cup \{N\}$,
I_r is the influencer set of observed models of r
$I_r \subseteq O \cup \{N\}, r \notin I_r$

$\forall o \in O \cup \{N\}$,
I_o is the influencer set of observed models of o
$I_o \subseteq R \cup \{N\}, o \notin I_o$

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 5 shows an example of connections between observed and observer models.

$\forall r \in R \cup \{N\}$,
$\forall d \in I_r, \exists$ an output translation function $Z_{r,d}$:
$Z_{r,d} : X_{obs?} \to X_{obs?_d}$, if $r = N$
$Z_{r,d} : Y_{obs?_r} \to Y_{obs?}$, if $d = N$
$Z_{r,d} : Y_{obs?_r} \to X_{obs?_d}$, if $r \neq N$ and $d \neq N$

$\forall o \in O \cup \{N\}$,
$\forall d \in I_o, \exists$ an output translation function $Z_{o,d}$:
$Z_{o,d} : X_{obs} \to X_{obs_d}$, if $o = N$
$Z_{o,d} : Y_{obs_o} \mapsto Y_{obs}$, if $d = N$
$Z_{o,d} : Y_{obs_o} \to X_{obs_d}$, if $o \neq N$ and $d \neq N$

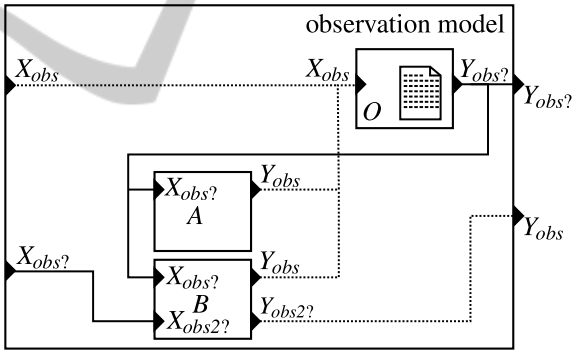

Figure 5: In this picture, we illustrate the second network in the coupled model to deal with observations graph. Plain connections are observation requests from output port $Y_{obs?}$ of observation models, to atomic model input port $X_{obs?}$ or to coupled model's output port $Y_{obs?}$, or from coupled 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.

2.2.4 Closure Under Coupling

Since we keep unchanged 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

Listing 1: Root coordinator algorithms.

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

  t = t0
  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
```

the coupled model), the PDEVS functions have the same content (See chapter 7 in (Zeigler et al., 2000) to the complete formalization). The closure under coupling property of the extension observation for parallel DEVS is still verified.

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.

As describe in the listing 1, the root coordinator is unchanged 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 $Z_{o,d}$.

The coordinator's algorithms are described in the listings 2. 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.

Listing 2: Coordinator algorithms.

```
1  Parallel−Devs−Coordinator
2  variables:
3    DEVN = (X, Y, D, {Md}, {Id}, {Zi,d})
4    parent // parent coordinator
5    tl // time of last event
6    tn // time of next event
7    eventlist // list of element (d, tnd) sorted by tnd
8    mail // output mail bag
9    mailobs // observation mail bag
10   Oevent // event−list of event observation model
11   yparent // output message bag to parent
12   yd // set of output message bags for each child d
13
14   when receive xobs−message (xobs,t)
15     if not (tl ≤ t ≤ tn) then
16       error: bad synchronisation
17     receivers = {o|o ∈ children, o ∈ Ir}
18     for each o ∈ receivers
19       send x−message (Zr,d(x) with input value Zr,d to o
20
21   when receive xobs?−message (xobs?,t)
22     if not (tl ≤ t ≤ tn) then
23       error: bad synchronisation
24     receivers = {r|r ∈ children, r ∈ Io}
25     for each r ∈ receivers
26       add r,xobs? to mailobs
27
28   when receive yobs−message (yobs,t) from o
29     for each child d ∈ Zo,d
30       send xobs−message to d
31     for each d ∈ Zo,d and d = N
32       send yobs−message to parent
33
34   when receive yobs?−message (yobs?,t) from r
35     for each child d ∈ Zr,d
36       send yobs−message to d
37     for each d ∈ Zr,d and d = N
38       send yobs?−message to parent
39
40   // finally, we append these algorithms
41   // in the following message
42
43   when receive i−message (i,t) at time t
44     [...] // The same algorithms than PDEVS
45     tb = tl
46
47   when receive ∗−message, x−message or y−mess
48   age
49     if tb != t then
50       for each r,xobs? ∈ mailobs
51         send xobs?−message (Zr,tb(x) with
52                                input value Zr,d to r
53       mailobs = ∅
54       tb = t
55     [...] // The same algorithms than PDEVS
56
57 end Parallel−Devs−Coordinator
```

Listing 3: Simulator algorithms.

```
Parallel−Devs−Simulator                                          1
variables:                                                       2
  parent // parent coordinator                                   3
  tl // time of last event                                       4
  tn // time of next event                                       5
  DEVS // associated model with total state (s,e)                6
  y_obs // output observation bag                                7
                                                                 8
  // The same algorithms than PDEVS simulator                    9
  // see chapter 11                                             10
  [...]                                                         11
                                                                12
  when receive xobs?−message (x_obs?, t) at t                   13
                              with input x_obs?                 14
    y_obs = λ_obs(x_obs?, t)                                    15
    send yobs−message (y_obs, t) to parent coordinator          16
                                                                17
end Parallel−Devs−Simulator                                     18
```

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).

The listing 3 describes the simulators algorithms. In line 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, we made an implementation of this extension in the DEVS open source simulator VLE.

3 RESULT

These 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 modeling. Therefore, VLE has evolved on a complete multi-modeling and simulation environment (Quesnel et al., 2009) and is now a generic environment for M&S, in Environmental Sciences, in Industry or Medicine. Many projects from two major French research institutes INRA and Cirad use VLE. For example, the RECORD project (Bergez et al., 2012) is a large platform designed for developing and analyzing models of cropping systems.

3.1 VLE: Virtual Laboratory Environment

VLE is a set of programs and libraries software. These components are used to model, to simulate and to analyze multimodel. VLE offers a set of formalism developed with a DEVS BUS (Kim and Kim, 1996). The sub-formalisms are DESS, QSS 1 and QSS 2 numerical procedure for solving ordinary differential equations, Petri nets, finite state automaton (moore, mealy, FDDEVS and Harel statechart (Harel, 1987)), cellular automaton (Cell-DEVS and Cell-QSS) and agent decision making. These extensions are developed using a DEVS BUS i.e. these extensions are developed such as atomic model. Finally, VLE uses an open-source license and all the source code are available on its website including this observation extension.

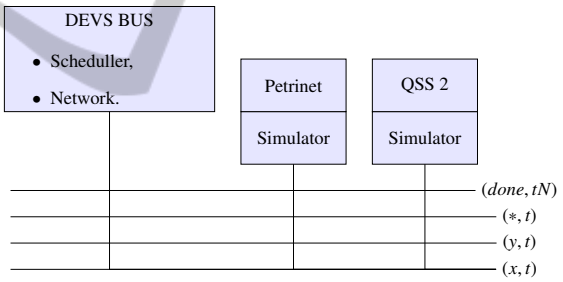

Figure 6: DEVS BUS and multi-modeling. The events (x,t), (y,t), $(*,t)$, $(done,tN)$ represent the bus.

3.2 Implementation

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

The listing 4 shows a simplified representation (without constructor, destructor, and other members) of the class which permits to develop atomic model behavior:

- l. 3-14: the classic PDEVS functions: ta, δ_{int}, δ_{ext}, λ and δ_{con}.
- l. 15: the observation method is called when an *observation event* arrived on a input observation

Listing 4: The C++ interface of the simulators in VLE.

```
class Dynamics {
public:
virtual Time timeAdvance() const;
virtual void internalTransition(
     const Time& time);
virtual void externalTransition(
     const Time& time,
     const ExternalEventList& lst);
virtual void output(
     const Time& time,
     ExternalEventList& out) const;
virtual void confluentTransition(
     const Time& time,
     const ExternalEventList& lst);
virtual Value* observation(
    const ObservationEvent& event) const;
};
```

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.).

Thus, the observation function of the simulator that implements the Euler method (a numerical procedure for solving ordinary differential equations) in VLE, the C++ code is:

```
Value* Euler::observation(
       const ObservationEvent& event) const
{
    double e = event.getTime() - mLastTime;

    return getEstimatedValue(e);
}
```

The function `getEstimatedValue` calculates an approximation of the value of the variable. It uses the current value of the state variable and the time elapsed since the last transition. This observation function does not change the current state of the variable. Thus, we respect the reproducibility of the simulation. However, we create additional cost calculations for each observation. This extra cost can be important if the observation function is complex and if called often. However, we believe that observing a model often is only needed when debugging the model.

3.3 Using Models with VLE

The study of the models is very important in the modeling and simulation cycle. Generally, this analysis is performed with an experimental setting. Experimental frames as sensitivity analysis, replicas generation are used to study or to validate models. One important motivation in formalizing the observation process in PDEVS is to develop generic experimental frames, which ones rely on intensive observation of models.

VLE is an environment for M&S. It relies on a set of libraries for modeling, simulation and analysis. These libraries developed in C++ are available for several programming languages. Thus, in order to collaborate with users of statistical tools, we provide an interface to the program R (R Development Core Team, 2012). R is a tool and language for statistical computing. This package named RVLE allows the user to run simulations, changes its parameters and retrieves simulation results (from the extension observation). This package combined with another packages of R as the classical *sensitity* package permit to build experimental frames to validate, explore and optimize models. On this principle, we also offer program and PYVLE JVLE to develop web services based on simulation.

4 CONCLUSIONS

In this paper, in section 2, we extend the PDEVS formalism to introduce the observation of models in the formalism. This work 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 building observations in a hierarchical and modular manner. The abstract simulators necessary for this extension were provided. In addition, we provided an implementation of this observation extension in the VLE simulators. 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 on the formalization of this new type of observation.

REFERENCES

Bergero, F. and Kofman, E. (2010). PowerDEVS: a tool for hybrid system modeling and real-time simulation. *SIMULATION*.

Bergez, J.-E., Chabrier, P., Gary, C., Jeuffroy, M., Makowski, D., Quesnel, G., Ramat, E., Raynal, H., Rousse, N., Wallach, D., Debaeke, P., Durand, P., Duru, M., Dury, J., Faverdin, P., Gascuel-Odoux, C., and Garcia, F. (2012). An open platform to build, evaluate and simulate integrated models of farming and

agro-ecosystems. *Environmental Modelling And Software*. In press.

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

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23:5–48.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8:231–274.

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

Kim, Y. J. and Kim, T. G. (1996). A heterogeneous distributed simulation framework based on DEVS formalism. In *Sixth Annual Conference On Artificial Intelligence, Simulation and Planning in High Autonomy Systems*, pages 116–121, La Jolla, California, USA.

Kofman, E. (2002). A second order approximation for devs simulation of continuous systems. *Simulation (Journal of The Society for Computer Simulation International*, 78(2):76–89.

Quesnel, G., Duboz, R., and Ramat, E. (2009). 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.

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

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

Vangheluwe, H. (2000). DEVS as a common denominator for hybrid systems modelling. In Varga, A., editor, *IEEE International Symposium on Computer-Aided Control System Design*, pages 129–134, Anchorage, Alaska. IEEE Computer Society Press.

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