multiPDEVS: A Parallel Multicomponent System Specification Formalism

Damien Foures[1,2], Romain Franceschini[1,*], Paul-Antoine Bisgambiglia[1], and Bernard P. Zeigler[3]

[1]CNRS UMR SPE 6134 - Université de Corse - Equipe SiSU
[2]CNRS; LAAS; Université de Toulouse; UPS, INSA, INP, ISAE; LAAS; F-31077 Toulouse, France
[3]RTSync Corp 12500 Park Potomac Ave. Potomac, MD, United-States of America
[*]Corresponding author: Romain Franceschini, r.franceschini@univ-corse.fr

Abstract

Based on multiDEVS formalism, we introduce multiPDEVS, a parallel and non-modular formalism for discrete event system specification. This formalism provides combined advantages of PDEVS and multiDEVS approaches, such as excellent simulation capabilities for simultaneously scheduled events, and components able to influence at each other using exclusively their state transitions. We next show the soundness of the formalism by giving a construction showing that any multiPDEVS model is equivalent to a PDEVS atomic model. We then present the simulation procedure associated, usually called *abstract simulator*. As a well-adapted formalism to express cellular automata, we finally propose to compare an implementation of multiPDEVS formalism with a more classical Cell-DEVS implementation through a fire spread application.

Index terms— System specification, PDEVS, discrete event simulation, multicomponent, multiDEVS, non-modular systems.

1 Introduction

An important concept of general system theory is that of decomposition, which allows a system to be broken down into smaller sub-systems to tackle its complexity, following a top-down approach. Conversely, interacting components may be coupled together resulting in a larger system following a bottom-up approach (Vangheluwe, 2008).

Zeigler et al. (2000) introduced the Theory of Modeling and Simulation (TMS), based on general system theory. Besides a framework which offers a precise description of the various entities involved in the modeling and simulation process, TMS provides a specification language for the modelling step called Discrete Event System Specification commonly known as DEVS. DEVS handles well modular and non-modular composition. With DEVS, modular systems interacts through I/O ports, whereas non-modular ones influences each other directly. While the former yields higher level of specification since it allows hierarchical construction, the latter remains interesting when modelling complex systems from bottom-up.

In order to facilitate even more the modeling process TMS has saw emergence of multiple DEVS based formalism. Most of them have been established for a modular and hierarchical approach, such as FD-DEVS for model with finite state space (Hwang and Zeigler, 2009), Cell-DEVS for cellular automata (Wainer and Giambiasi, 2001), RT-DEVS for real-time modelling (Cho and Kim, 1998), ST-DEVS for stochastic modelling (Kofman and Castro, 2006). For non-modular approach, B.P. Zeigler introduced another DEVS based formalism, named multiDEVS (Zeigler et al., 2000).

Despite good extensibility properties, classic DEVS formalism has its own limitations. As with all other event scheduling approaches, it is up to the modeler to understand interactions between models and manage collisions that can occurs between events. Consequently, the behavior of the model can easily deviate from the expected one if such collisions are not properly managed (Chow and Zeigler, 1994). This issue prompted A.C.-H. Chow to propose PDEVS, which includes entities dedicated to event collision handling, but only for the modular approach.

In this context, we propose a new formalism bringing effective management of event conflicts as was initially proposed within PDEVS, combined with the multiDEVS modeling approach. We propose a construction of equivalence with the PDEVS formalism and provide the associated abstract simulator. We call this formalism multiPDEVS.

multiPDEVS was initially created for a specific work, where an existing set of PDEVS models had to be coupled with non-modular approach. However, this paper is fully dedicated to the formalism itself, and a simple example to help each knowledge level, from beginner to the expert in DEVS formalisms. This formalism is well adapted to problem classes similar to cellular automata (Al-Habashna and Wainer, 2016; Wainer and Castro, 2010), and could be used in problems such as circulation management, robot path planning, physical propagation or crowd modeling. Unlike specialized formalisms such as Cell-DEVS (Wainer and Giambiasi, 2001) where functions and structures are integrated to facilitate the specification process (e.g. the delay function of Cell-DEVS formalism), multiPDEVS does not integrate such specificities, which means that multiPDEVS is intended to be more generic and could be adapted to other classes of problems.

The next section gives a review of the PDEVS and multiDEVS formalisms where limitations and advantages of such formalisms are discussed. Section 2 presents recent applications of the multicomponent approach. Then, a third section gives a detailed description of the multiPDEVS formalism. Section 4 describes an example to illustrate the use of the formalism. Section 5 gives a clear overview of pros and cons of multiPDEVS from a modelling perspective, but also from a simulation perspective. The last section gives conclusions and perspectives.

2 Related works

As far as we know, there are few works which explicitly use the multiDEVS formalism despite decades of existence (Zeigler et al., 2000).

We can cite Innocenti et al. (2004) and Muzy et al. (2003) where the non-modular multicomponent is explicitly used. Based on multiDTSS (multicomponent Discrete Time System Specification) (Zeigler et al., 2000), the authors extend this formalism with the *Active* function (*AmultiDTSS*) to focus on activated cells of cellular forest fire propagation models in order to increase simulation performances. They agree that a non-modular multicomponent based approach helps to reduce modeling and simulation complexity for cellular systems compared to a conventional modular approach as proposed by the DEVS formalism. Indeed, the feature of influencer/influencee offered by the multicomponent approach allow to avoid many messages exchanges, which permit to increase the simulation speed.

We can observe an equivalent message reduction principle within the flattening process of DEVS models where simulation speed improvement have been studied (Bae et al., 2016; Jafer and Wainer, 2009; Zacharewicz et al., 2010). However, the flattening process must to be clearly distinguished from multicomponent modeling. The flattening process of PDEVS model can be considered as a modelling *transformation*, where multicomponent approach must be considered as a modelling *process*. As given by Chen and Vangheluwe (2010), the flattening of DEVS model consist of two steps: *direct connection* step where the coupled model is transformed to a coupled model of depth one, and *flattening* step where the coupled model of depth one is transformed in an atomic model, for the sole purpose of improving simulation performance. As we will see in details subsequently, multiPDEVS is dedicated to non-modular modelling process.

As introduced in the previous section, we believe that the multiDEVS formalism as initially proposed does not allow proper management of conflicts. Recent works of Shiginah and Zeigler (2011) confirms our idea that the multicomponent approach is an interesting modeling approach but somewhat neglected due to lack of efficient conflict management system. The authors offer a new specification for cellular DEVS models to increase performances of cell space simulations (Shiginah and Zeigler, 2011). As a perspective, the authors recall an alternative consisting in the modification of multiDEVS, such that it becomes equivalent to the PDEVS formalism.

In this section, we make a review of necessary background, namely multiDEVS formalism and PDEVS formalism as introduced by Zeigler et al. (2000). We also briefly introduce the new version of CellSpace DEVS formalism as proposed by Shiginah and Zeigler (2011) to give the opportunity to the reader to understand the advantages of both formalisms.

2.1 The original multicomponent DEVS formalism

There are two types of DEVS models, atomic and coupled. An atomic model describes the behavior of a system, while a coupled model describes a system as a network of components coupled together. These models are in the modular form, which means their interactions are restricted to happen through identified ports whose connection is pre-established. The multiDEVS formalism, which is based on classic DEVS, introduce a non-modular way to couple components together where components can interact with each other by accessing and influence other components states directly via state transition functions. As an illustration, we can take the cellular automaton example as proposed by Muzy et al. (2003):

In a non-modular cellular automaton, simple neighboring rules can be implemented for every cell as: If my neighboring cell is alive, then I become alive. In a modular cellular automaton, the specification is different. It would be: If the message received from my neighboring cell indicates that it is alive, then I become alive.

For readers already acquainted with classic DEVS mechanism (but not necessarily with multicomponent approach), we should warn that the multiDEVS formalism can look forbidding at a first look.

In a multicomponent system, a component has its own set of states and state transition functions. Each component also has a set of *influencers* components from which it may be influenced and a set of *influencees* that it may influence through its state transition functions. Those interacting components form the overall system. An input to the system may influence all components and each component may contribute to the output of the system. Components are not considered stand-alone system specifications since they are devoid of an input and output interface.

In the hierarchy of system specifications, the multicomponent system specification lies in a lower level than a modular coupled network of systems such as a DEVS coupled model. Although both specifications allows the composition of interacting components to form a new system, components in the former are coupled non-modularly since they influence each other directly while in a coupled network of systems, components modularly interacts only through their I/O interfaces. If a modular approach is considered a more abstract level of specification, particular models of systems may be more suitable to be described in a non-modular way. Systems such as cellular automata, individual-based models or some kinds of multi-agent systems environments (cellular models) might be good examples. In contrast, systems with structural constraints that may be encountered in systems engineering will be best modelled using a modular compositional approach. We should emphasize that both approaches may exist within the same modelled system. Indeed, a multicomponent DEVS system can be coupled together with other DEVS models. Another advantage using multicomponent DEVS is that since components do not interacts through I/O interfaces, the implementation cost of message routing is reduced leading to better performances (Bae et al., 2016; Jafer and Wainer, 2009).

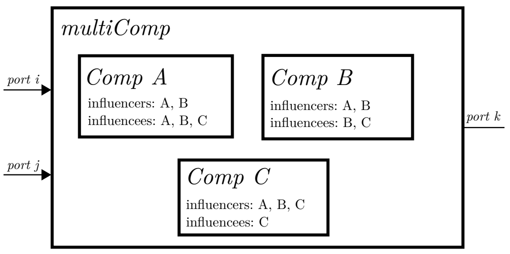

Figure 1: A multiDEVS model: components can interact through state transition functions.

Figure 1 illustrates this principle with a simple case using three components: $CompA, CompB, CompC$. In this example, $CompA$, $CompB$ and $CompC$ will have the opportunity to directly change the state of $CompC$ through its own state transition functions ($CompA$, $CompB$, $CompC$ are considered as the set of influencers of $CompC$). Notice that there is no reciprocity between influencers and influencees. That means it is essential to make the distinction between them. Let's consider the relation between $CompA$ and $CompB$ (Figure 1). $CompA$ declares $CompB$ as an influencer whereas $CompB$ do not mention $CompA$ as one of these possible influencees. This means that $CompA$ can consult $CompB$ states to take decisions, but $CompB$ is not able to directly change the state of $CompA$. Note that components could have direct interactions with interfaces of the system ($port\ i, port\ j, port\ k$) through specific functions (see δ_{ext} and λ in following paragraph).

A multicomponent DEVS is a structure:

$$multiDEVS = (X, Y, D, \{M_d\}, Select)$$

where X and Y are the input and output event sets such as:

$$X = \{(p,v) \mid p \in IPorts, v \in X_p\}$$
$$Y = \{(p,v) \mid p \in OPorts, v \in Y_p\}$$

with X_p the set of all possible values for the input port p and Y_p the set of all possible values for the output port p.

D is the set of component references and $Select$ is a tie-breaking function employed to arbitrate in case of simultaneous events:

$$Select : 2^D \to D.$$

For each $d \in D$, component M_d is defined by

$$M_d = (S_d, I_d, E_d, \delta_{ext,d}, \delta_{int,d}, \lambda_d, ta_d)$$

where S_d is the set of sequential states of d, $I_d \subseteq D$ is the set of components influencing d, $E_d \subseteq D$ is the set of influenced components by d. A pair (s,e) represents a total state that includes e the time elapsed since the last transition and Q_d is the set of total states:

$$Q_d = \{(s,e_d) \mid s \in S_d, e_d \in \mathbb{R}_0^+, 0 \le e \le ta(s)\}.$$

Each component $d \in D$ of a multiDEVS schedule the time of its next internal event based on total states of its influencers using the time advance function:

$$ta_d : \underset{i \in I_d}{\times} Q_i \to \mathbb{R}_0^+ \cup \{\infty\}.$$

When an internal event occurs in one of the components, state changes occurs through the internal transition function that takes the total state set of the influencers and maps them into new total states for the influencees:

$$\delta_{int,d} : \underset{i \in I_d}{\times} Q_i \to \underset{j \in E_d}{\times} Q_j.$$

That means that an internal event occurring in d is able to change states of other components and may result in rescheduling their events. If this behavior is not desired because a component should only be allowed to change its own state, then E_d should be defined as a unit set whose unique element is d.

Eventually, output events may be generated for the multiDEVS through the output function:

$$\lambda_d : \underset{i \in I_d}{\times} Q_i \to Y.$$

The output of the multiDEVS is defined by the output event of the component selected over imminent components. When events in different components are imminent, the *Select* function is used to arbitrate among them. External events at the multiDEVS's input interface can be handled by any of the external state transition function $\delta_{ext,d}$ of the component d:

$$\delta_{ext,d} : \underset{i \in I_d}{\times} Q_i \times X \to \underset{j \in E_d}{\times} Q_j.$$

However, a component may not react to external inputs if its external state transition function is not defined and, similarly, λ_d can be left undefined if the component is not expected to produce output events.

As Chow and Zeigler (1994) stated, when a DEVS model is constructed by coupling components together, such model behavior may deviate from the expected one since multiple state transitions can happen at the same simulation time. Two kind of such transition collisions are distinguished:

- *collisions*, which occur when a scheduled internal event overlaps with one or more external events;
- and *simultaneous events*, which occur when several external events occur at the same time.

The Classic DEVS and multicomponent DEVS solutions to transition collisions are not satisfactory since the behavior of a coupled model is established through the tie-breaker *Select* function. Concerning collisions as defined above, the priority of internal versus external events depends on the priority imposed by the *Select* function over imminent components (components with simultaneous scheduled internal events). Among those imminent components, the internal event is always preferred to the external ones for the first favored component. For the remaining components, external events might occur before the internal one since the output event of a first activated component always take over the internal event of another imminent component. Regarding simultaneous events, the order at which one arrive is also a result of the priority given by *Select*. For the same simulation time, the external transition is sequentially activated each time an external event arrives. Consequently, if a state depends on two external events, the modeler must consider a transitory state. DEVS imposed serialization solution to transition collisions we just described is a weakness because it can be extremely difficult for the modeler to choose the sequence of internal events corresponding to the overall expected behavior of the model, especially if there is a lot of mutually influencing components. Furthermore, it prevents taking advantage of event simultaneity by parallelizing their process. For those reasons, Chow and Zeigler (1994) introduced the Parallel DEVS (PDEVS) specification which facilitate handling both kind of transition collisions.

2.2 Parallel DEVS

In contrast to the classical DEVS formalism, the parallel form, namely PDEVS (Chow and Zeigler, 1994), allows the modeler to properly handle collisions and simultaneous events at the component level. PDEVS makes several changes to the structures of atomic models and coupled models, as well as the abstract simulators. A demonstration that PDEVS preserves the closure under coupling property is given in Chow and Zeigler (1994) by constructing the resultant of a coupled model as a well-defined PDEVS atomic model.

In order to provide a solution to transition collisions that might occur during a simulation (*cf.* section 2.1), PDEVS suggest the following structure for an atomic model:

$$PDEVS = (X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$$

where X and Y are the input and output event sets, and S is the set of sequential states as defined in the multiDEVS specification. The pair (s,e) form a total state where e represent the time elapsed since the last transition and Q is the set of total states:

$$Q = \{(s,e) \mid s \in S, e \in \mathbb{R}_0^+, 0 \leq e \leq ta(s)\}.$$

The time advance function $ta(s)$ determines the interval during which the current state remains valid, unless an external event occurs:

$$ta : S \rightarrow \mathbb{R}_{0,\infty}^+.$$

When the time elapsed $e = ta(s)$, the current state expires. Before calculating its new state, the model may produce a set of outputs for the current state through the output function:

$$\lambda : S \rightarrow Y^b.$$

A new state is then calculated using the internal transition function:

$$\delta_{int} : S \rightarrow S.$$

However, an external event that occurs on any input port will beget a new state calculated using the external transition function:

$$\delta_{ext} : Q \times X^b \rightarrow S.$$

In contrast to DEVS, the δ_{ext} function is given a bag of input events instead of a single input event. This modification allows the modeler to properly handle *simultaneous events* we discussed in section 2.1, since all simultaneous inputs intended for this model are available in a single transition.

In order to handle *collisions* (*cf.* section 2.1), PDEVS introduce the confluent transition function δ_{con}, allowing to explicitly manage them rather than serializing model behavior at collision times:

$$\delta_{con} : S \times X^b \rightarrow S.$$

The modeler has complete control if any collision is to occur. He can choose what serialization to use (either the sequential activation of δ_{int} and of δ_{ext} or the other way around) or a specific behavior.

The DEVS and PDEVS formalisms allows to describe systems in a modular and hierarchical way as a network of coupled components, where components are atomic or coupled models. PDEVS defines such network as:

$$N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,j}\})$$

where X and Y are the input and output event sets, D is the set of component references and for each $d \in D$, M_d is a PDEVS atomic model as defined above or another coupled model as defined here.

Couplings between components are represented with the help of a set I_d, where for each $d \in D \cup \{N\}$, I_d represent the set of components *influencing* d, formally defined as:

$$I_d \subseteq D \cup \{N\} \qquad \text{with } d \notin I_d$$

Along with I_d, couplings are given by the output to input function $Z_{i,d}$. Thus, for each $d \in D \cup \{N\}$ and for each $i \in I_d$:

$$\begin{aligned} Z_{i,d} &: X_N \rightarrow X_d && \text{if } i = N \\ Z_{i,d} &: Y_i \rightarrow Y_N && \text{if } d = N \\ Z_{i,d} &: Y_i \rightarrow X_d && \text{if } i \neq N \text{ and } d \neq N \end{aligned}$$

The PDEVS coupled model structure is near identical to a DEVS coupled model. Only the tie-breaker *Select* function which originally allows the modeler to prioritize an atomic model among the set of imminent components ($\min\{ta_i(s_i \mid i \in D)\}$) disappears. In PDEVS semantics, the activation of components scheduled at the same simulation time is not serialized as in DEVS, so *Select* becomes unnecessary. Instead, a two-phase approach is employed: (1) collect all imminent components outputs; and (2), perform appropriate transitions which are *internal* transitions for all imminent components, *external* transitions for components about to receive inputs given the collected outputs in the first phase and *confluent* transitions for imminent components which also receives inputs.

In addition to providing a specification which facilitate handling transition collisions, PDEVS semantics benefits from the intrinsic parallelism that imminent components in a network of models offers. Each time an internal event occur, each phase of the simulation protocol as described above can be easily parallelized among components, with a sync barrier between the two phases.

2.3 CellSpace DEVS specification

As introduced previously, Shiginah and Zeigler (2011) present a new cell space DEVS specification for faster model development and simulation efficiency based on their previous work (Shiginah, 2006). This new specification offers a significant improvement in simulation speed by providing effective management of active cells and a strong limitation of the number of inter-cell messages.

Usually, a cellular system is described using an atomic DEVS model for each cell (cell-DEVS (Wainer and Giambiasi, 2001)). To reduce inter-cell messages, authors integrate all the cells in a single atomic model called *atomic CellSpace*. To get there, and it is here where our works meet, authors transform the modular description of the system to a non-modular version using a restricted version of the influencer/influencee principle described in multiDEVS (cf. section 2.1). In contrast to multiDEVS, they decide to protect the cell integrity and do not allow to write the state variables of other cells (read only). This is expressed in our formalism as a restriction of all influenced components (E_d) of a component d to the component itself. Formally, E_d is restricted to the singleton $\{d\}$, $E_d = \{d\}$. Perfectly acceptable for their application to cellular automata, this restriction allows authors to easily manage state changes in cells with a classical storage of the current state of all cells. In our case, the proposed formalism aims to be more generic and this restriction is no longer suitable. This will force us to integrate a new state collision management mechanism, as presented in the next section.

3 A parallel multicomponent DEVS formalism

To our knowledge, the multicomponent approach as defined by Zeigler et al. (2000) never benefited from the refinements provided by the PDEVS formalism concerning transition collisions management. We propose an attempt to bring those refinements to the multicomponent approach along with a way to handle another kind of collisions, specific to the non-modular approach, which we call *state collisions*. Such state collisions simply happen when multiple transitions occur at the same simulation time. They are related to the way components interacts in multicomponent systems (directly changing state variables of other components through state transitions).

Because components are allowed to access and write on each other's state, we can consider each transition as a violation of other components autonomy in the sense that they don't evolve having full control over their states and thus, their behavior. In contrast, a PDEVS atomic model ensures autonomy through modularity because the only way to influence it externally is through an external event. We believe this property is essential for proper modelling of a given system and should be taken in consideration for the multicomponent approach.

In the multiDEVS abstract simulator (Zeigler et al., 2000), as we explained in section 2.1, all imminent components scheduled for the next simulation time are serialized and prioritized via the *Select* function. As with DEVS, for the same simulation time, the order at which components are activated may produce a behavior that is not the intended one in the first place and that is difficult to tackle down. As an example, consider a multicomponent system composed of three components A, B and C as illustrated in Figure 1. Figure 2 illustrate a particular scenario where the ta of components A and B is equal to 0, which means an internal event is due for both of them at the same simulation time (step (a)). It means that the internal transition of both components (1) and (2) will sequentially be activated following the priority given at the multicomponent level by the *Select* function (step (b)). Here, priority is first given to component A, which means it may write directly on the state of component C. Consequences of this particular choice may result in different possible scenarios from here. The state change made in C could result in another internal event due for the same simulation time (in that case the two imminent components remaining are B and C). Another possibility is that B remains the only imminent component and its internal transition is finally activated (which also may write on the state of C, and could lead B to overwrite changes made by A on C state). In our opinion, there is two kind of difficulties modelling such system: the behavior becomes quickly difficult to anticipate and direct changes on components states are conceptually arguable.

Recall semantics of PDEVS, which carries outputs of all imminent components, and then all appropriate transitions for the same simulation time. In order to apply PDEVS semantics to the multicomponent approach, we have to apply the same solutions concerning collisions and simultaneous events as long as a solution to state collisions. Simultaneous transitions in a multicomponent system may imply multiple states generated for a given component at the same simulation time. Rather than serializing those states, we take the same path as PDEVS that is leaving the decision to the modeler. This way, one can decide which state change to apply in a particular order or how to compose those multiple states into a new one. Not only it allows to solve state collisions but also provides a way to introduce component autonomy in non-modular systems: the component is fully responsible for its state at any moment.

In this section, we present our proposal of a formal multicomponent approach using PDEVS semantics as

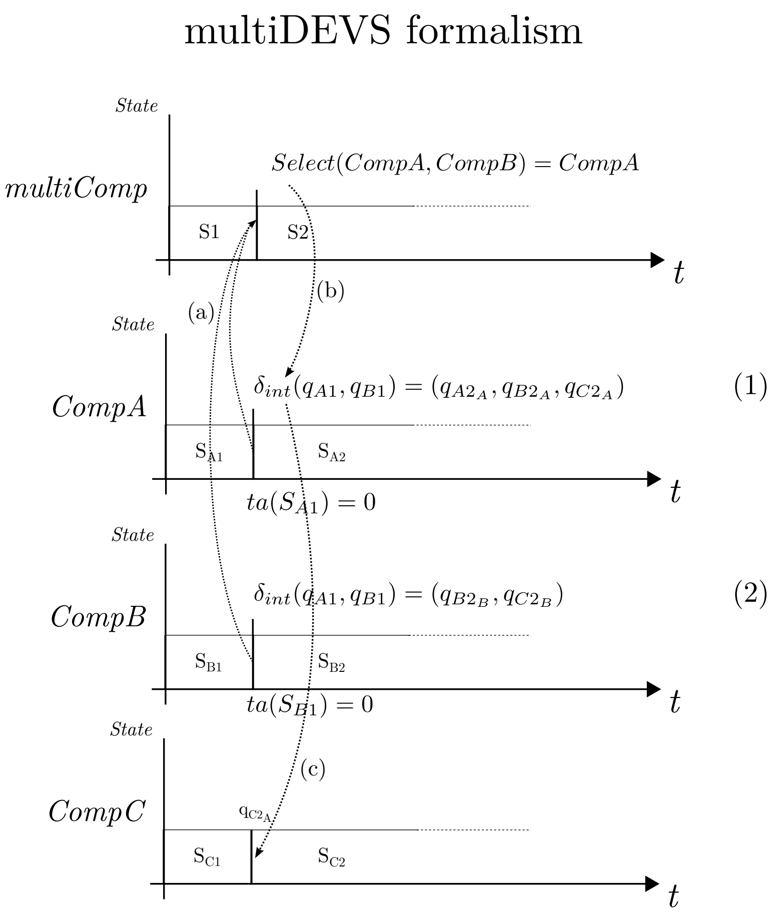

Figure 2: Illustration of state collision serialization using multiDEVS.

introduced above, called *multiPDEVS*. We also present the construction of equivalence of multiPDEVS with a PDEVS atomic model and finally, the abstract simulator following PDEVS protocol is given.

3.1 multiPDEVS

In order to propose a parallel multicomponent system specification, we argued in the above discussion that we need to apply PDEVS semantics to the non-modular approach alongside a way to manage state collisions.

In the multicomponent DEVS specification, components are not provided with their own I/O interface. Still, they remain able to communicate with other modular components via the interface of their parent, the multiPDEVS itself. When an external event occurs at the multiPDEVS level, all components that defined an external transition function receive this event. Similarly, all components that defined an output function are able to produce outputs at the multiPDEVS level. To be equivalent with PDEVS, we introduce in the component structure the set of bags of inputs X^b over elements in X allowing the modeler to handle simultaneous events and also introduce the confluent transition function δ_{con}, allowing the modeler to explicitly define the collision behavior. To manage state collisions, we propose the reaction transition function, δ_{reac}, which gives the modeler a chance to explicitly define state collisions behavior. That implies to modify others state transition functions, so that δ_{int}, δ_{ext} and δ_{con} do not give new states to influencees, but suggest them instead. The reaction transition is given as an argument a bag whose elements includes all states suggested by components able to influence this particular component for the same simulation time. Each suggested state is accompanied with the identity of the component that produced it. We note K^b the set of bag of proposed partial states over elements in K.

The structure of the multicomponent parallel DEVS is:

$$multiPDEVS = (X, Y, D, \{M_d\})$$

where X, Y, and D are defined as in multiDEVS. Note that the *Select* function, as in PDEVS, disappears from the multiPDEVS definition. It becomes useless since all imminent components are handled simultaneously rather than being serialized.

For each $d \in D$, a component M_d is defined by the structure:

$$M_d = (S_d, I_d, E_d, \delta_{ext,d}, \delta_{int,d}, \delta_{con,d}, \delta_{reac,d}, \lambda_d, ta_d)$$

where S_d is the set of sequential states of d, $Q_d = \{(s, e_d) \mid s \in S_d, e_d \in \mathbb{R}_0^+, 0 \le e_d \le ta_d(s_d)\}$ is the set of total states with e the time elapsed since the last transition, $I_d \subseteq D$ is the set of influencing components and $E_d \subseteq D$ is the set of influenced components. Components schedule the time of their next internal event based on total states of their influencers using the time advance function:

$$ta_d : \underset{i \in I_d}{\times} Q_i \rightarrow \mathbb{R}_0^+ \cup \{\infty\}.$$

When $e_d = ta_d(s)$, the component may generate a set of outputs for the multiPDEVS output interface via the output function (which can be left undefined):

$$\lambda_d : \underset{i \in I_d}{\times} Q_i \rightarrow Y^b.$$

Then, the internal event being due, its internal transition function is activated. It takes the total state set of the influencers and maps them into suggested states for the set of influencees:

$$\delta_{int,d} : \underset{i \in I_d}{\times} Q_i \rightarrow \underset{j \in E_d}{\times} S_j$$

If one or several new states are produced through state transitions of components able to influence d, then the d reaction transition function is activated in order to let the modeler explicitly define the state collision behavior. For instance, if two components suggest each a new state for component d where a particular variable is incremented, then d should be able to produce a new state where the variable value depends on the two suggested suggested states and the current state of d. To allow this, the function is given the bag of suggested states K_d^b and the current state of the component:

$$\delta_{reac,d} : K_d^b \times Q_d \rightarrow S_d$$

where

$$K_d = \{(s_d, c) \mid s_d \in S_d, d \in E_c\}.$$

The tuple (s_d, c) is a suggested state for d produced by component c, where $d \in E_c$, and the set of bag of suggested states for d over elements in K_d is noted K_d^b.

This function takes the bag of states produced by other components including d in their set of influencees alongside its current total state in order to produce its new valid total state. The reason we use only Q_d instead of the cross-product of all influencers states ($\underset{i \in I_d}{\times} Q_i$) is to ensure coherence of states. If we allow this, we have to ensure that all reads on $I_d - \{d\}$ are performed on current states and not on new ones produced by their own δ_{reac} function, which can happen if proposed states have been produced at the same simulation time by components for components included in $I_d - \{d\}$.

When external events occur at the multiPDEVS level, its components may receive them if they define their corresponding external transition function:

$$\delta_{ext,d} : \underset{i \in I_d}{\times} Q_i \times X^b \rightarrow \underset{j \in E_d}{\times} S_j.$$

As in PDEVS (in contrast to DEVS and multiDEVS), $\delta_{ext,d}$ is given a bag of inputs that contains all simultaneous external events instead of a single input event. If an internal event is due simultaneously with one or several external events, a collision is to be managed. The confluent transition function, originally introduced in PDEVS, allows to explicitly manage the collision behavior:

$$\delta_{con,d} : \underset{i \in I_d}{\times} Q_i \times X^b \rightarrow \underset{j \in E_d}{\times} S_j.$$

Note that, similarly to the internal transition function, $\delta_{ext,d}$ and $\delta_{con,d}$ generate a set of proposed states (S) for the influencees instead of directly update their states.

The semantics of multiPDEVS are illustrated in Figure 3 using the same scenario we presented in section 3 with Figure 2. Components A and B are imminent. Both of them perform their internal state transition ((1) and (2)) generating simultaneously new states for component C, which is a state collision (step (a)). C is given complete control over its future state via the δ_{reac} function, whose behavior may consist in choosing what state to apply or composing a new one (step (b)).

A multiPDEVS works in the following way. As in PDEVS, the activation of imminent components is done simultaneously using a two-phase approach. In the first phase, the outputs of all components that defined a λ function are collected. We divide the second phase in three other micro-steps: in the first micro-step, appropriate state transitions are performed and their outputs (the state bags) are temporarily saved for the second micro-step. For components that defined δ_{ext}, external transitions are activated when external events occur on the input interface of the multiPDEVS. For all imminent components, when there is also inputs available at the multiPDEVS level, confluent transitions are activated for components who defined δ_{ext}, otherwise their δ_{int} is activated. If there are no inputs, internal transitions are activated for all imminent components.

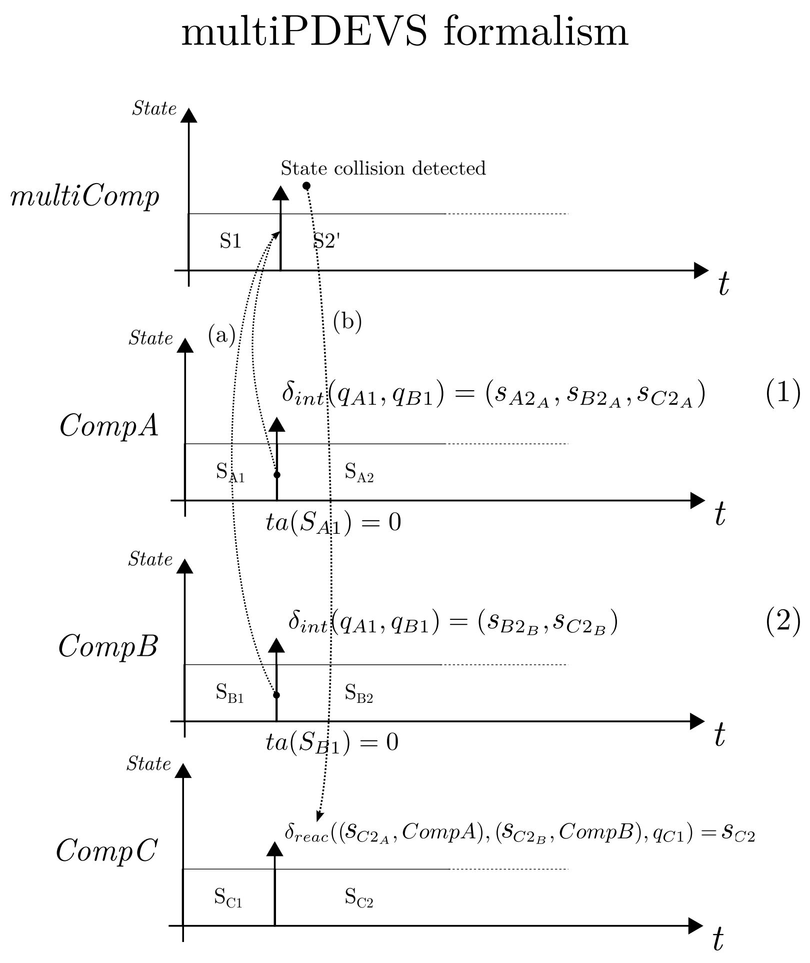
multiPDEVS formalism
State
State collision detected
multiComp
S1
S2'
t
(a)
(b)
State
$\delta_{int}(q_{A1}, q_{B1}) = (s_{A2_A}, s_{B2_A}, s_{C2_A})$ (1)
CompA
S_{A1}
S_{A2}
$ta(S_{A1}) = 0$
State
$\delta_{int}(q_{A1}, q_{B1}) = (s_{B2_B}, s_{C2_B})$ (2)
CompB
S_{B1}
S_{B2}
$ta(S_{B1}) = 0$
State
$\delta_{reac}((s_{C2_A}, CompA), (s_{C2_B}, CompB), q_{C1}) = s_{C2}$
CompC
S_{C1}
S_{C2}

Figure 3: Illustration of state collision management using multiPDEVS.

The second micro-step consists of activating the reaction transition function for all components that have a bag of states generated during the first micro-step. Finally, the third micro-step consist in calculating the time of next events of each influenced component using the time advance function. Such semantics can be integrated in a PDEVS atomic model, which allow to integrate multiPDEVS in a larger PDEVS simulation.

3.2 Construction of equivalence with PDEVS

In this section, we give proof of multiPDEVS equivalence with PDEVS by constructing the multiPDEVS resultant as a well-defined atomic PDEVS. This construction ensures that a multiPDEVS model may be coupled with other PDEVS models within a coupled model.

A multiPDEVS defines a PDEVS as follows.

Given a multiPDEVS as defined earlier, we associate :

$$PDEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$$

where $S = \underset{d \in D}{\times} Q_d$.

We define

$$ta(s) = min\{\sigma_d \mid d \in D\}$$

with $\sigma_d = ta_d(\ldots,(s_i,e_i),\ldots) - e_d$ as the remaining time until the next scheduled event in d.

We define the set of imminent components as:

$$IMM(s) = \{d_\sharp \mid d_\sharp \in D, \sigma_{d_\sharp} = ta(s)\}.$$

We define for each $d \in D$, $S_d = \{s_d \mid d \in E_{d_\sharp}\}$, the set of suggested states produced by $d_\sharp$ and $K_d = \{(s_d, d_\sharp) \mid d_\sharp \in IMM(s), d \in E_{d_\sharp}\}$, the set of suggested states and their producers for each influenced component d.

Given that $d_\sharp \in IMM(s)$ and

$$\delta_{int}(q_1, q_2, \ldots, q_n) = (q'_1, q'_2, \ldots, q'_n)$$

with

$$(s'_j, e'_j) = \begin{cases} (s_j, e_j + ta(s)) & \text{if } k^b_j = \varnothing \\ (\delta_{reac,j}(k^b_j, (s_j, e_j + ta(s))), 0) & \text{otherwise} \end{cases}$$

where

$$k_j^b = \{(\delta_{int,d_\sharp}(\dots,(s_i,e_i+ta(s)),\dots)\bullet j,d_\sharp) \mid d_\sharp \in IMM(s), j \in E_{d_\sharp}, i \in I_j\}$$

The resultant internal transition function contains two kinds of component transitions. For components influenced by imminent ones, $\delta_{reac,j}(k_j^b,(s_j,e_j))$ function is called. The state transition function $\delta_{int,d_\sharp}$ of each imminent $d_\sharp$ is executed, where $(\dots,(s_i,e_i+ta(s)),\dots)$ is the state vector of the influencing components $i \in I_{d_\sharp}$, which produces a set of proposed states for all influenced components $j \in E_{d_\sharp}$. In order to fill the incoming bag of proposed states (k_j^b) of each $j \in E_{d_\sharp}$, each suggested state produced for j will result in a tuple composed of the state and its corresponding producer. For components having their bag of proposed states empty ($k_j^b = \varnothing$), the elapsed time is simply updated.

The output of the system is defined by

$$\lambda(s) = \{\lambda_{d_\sharp}(\dots,(s_i,e_i+ta(s)),\dots) \mid d_\sharp \in IMM(s), i \in I_{d_\sharp}\}$$

The external transition function δ_{ext} upon occurrence of an input bag of events x^b is defined by the cross product of the external state transition functions of all components $d \in D$. We define the overall external transition function by

$$\delta_{ext}((q_1,q_2,\dots,q_n),e,x^b) = (q_1',q_2',\dots,q_n')$$

with

$$(s_j',e_j') = \begin{cases} (s_j,e_j+e) & \text{if } k_j^b = \varnothing \\ (\delta_{reac,j}(k_j^b,(s_j,e_j+e)),0) & \text{otherwise} \end{cases}$$

where

$$k_j^b = \{(\delta_{ext,d}((\dots,(s_i,e_i+e),\dots),x^b)\bullet j,d) \mid d \in D, i \in I_d, j \in E_d, \exists \delta_{ext,d}\}$$

With a similar mechanism, the incoming bag of proposed states (k_j^b) of each influenced component is built using $\delta_{ext,d}$ function. As previously, other components where the bag of proposed states is an empty set ($k_j^b = \varnothing$), the elapsed time is simply updated.

The confluent transition function δ_{con} is defined by:

$$\delta_{con}((q_1,q_2,\dots,q_n),x^b) = (q_1',q_2',\dots,q_n')$$

with

$$(s_j',e_j') = \begin{cases} (s_j,e_j+ta(s)) & \text{if } k_j^b = \varnothing \\ (\delta_{reac,j}(k_j^b,(s_j,e_j+ta(s))),0) & \text{otherwise} \end{cases}$$

where

$$k_j^b = \begin{cases} \{(\delta_{int,d}(\dots,(s_i,e_i+ta(s)),\dots)\bullet j,d) \\ \quad \mid d \in IMM(s), i \in I_d, j \in E_d\} \\ \qquad \text{if } \nexists \delta_{ext,d}, d \in IMM(s) \\ \{(\delta_{con,d}((\dots,(s_i,e_i+ta(s)),\dots),x^b)\bullet j,d) \\ \quad \mid d \in IMM(s), i \in I_d, j \in E_d\} \\ \qquad \text{if } \exists \delta_{ext,d}, d \in IMM(s) \\ \{(\delta_{ext,d}((\dots,(s_i,e_i+ta(s)),\dots),x^b)\bullet j,d) \\ \quad \mid d \notin IMM(s), i \in I_d, j \in E_d\} \\ \qquad \text{if } \exists \delta_{ext,d}, d \notin IMM(s) \end{cases}$$

Finally, the incoming bag of proposed states (k_j^b) dedicated to the resultant confluent transition function is composed using three different contributions: from imminent components where $\delta_{ext,d}$ function is undefined (influence comes from $\delta_{int,d}$), from imminent components where a $\delta_{ext,d}$ function is defined (influence comes from $\delta_{con,d}$), and finally from non-imminent components where a $\delta_{ext,d}$ function is defined (influence comes from $\delta_{ext,d}$).

Note that such proof ensures that multiPDEVS models can be transformed to atomic PDEVS models and used as-is with a PDEVS simulator. A preferable solution is to avoid any transformation of the multiPDEVS model, using the DEVS bus principle to simulate the multiPDEVS in its original form, which we detail in the following section.

3.3 multiPDEVS abstract simulator

Among the benefits of DEVS formalisms, model/simulator distinction is one of the key elements. This leads to a clear separation of concerns. It helps to dissociate what is related to the model, from what is related to how it is executed. This makes DEVS models easier to reuse and exchange (Zeigler and Sarjoughian, 2013).

In order to integrate the multiPDEVS approach into a PDEVS-based simulation environment, we use the DEVS bus concept by wrapping multiPDEVS into a PDEVS form along with its own dedicated simulator. This allows multiPDEVS models to be modularly coupled with other models and executed through a PDEVS coordinator. The multiPDEVS formalism presented in section 3.1 is compatible as-is with a PDEVS since we provide an I/O interface (ports at the multiPDEVS level) and associated functions (δ_{ext}, δ_{con} and λ at the component level). The simulator we define here sticks to the simulation mechanism of PDEVS by following its communication protocol as defined in Chow et al. (1994) as illustrated in Figure 4.

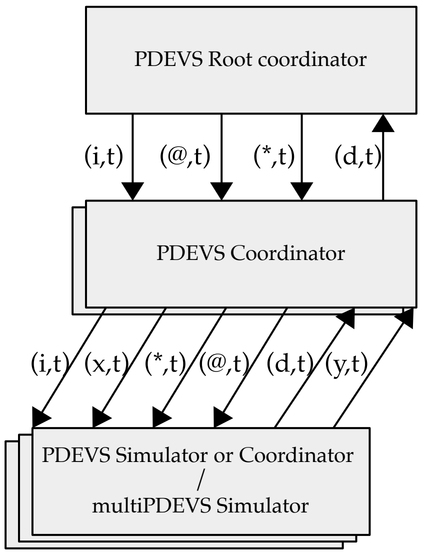

Figure 4: multiPDEVS abstract simulator integrated within PDEVS simulation protocol.

Abstract simulators, or simulation processors, associated with PDEVS models comes in two forms, *simulator* and *coordinator*. While a simulator role is to execute the atomic model functions, a coordinator role is to carry out and manage output events of its children as long as their scheduling.

We intentionally omit the *root coordinator*, but we note that it oversees the whole simulation process and initializes the simulation time (t) with the min tn of its children.

The simulator we present in the following assumes both simulator and coordinator roles since it realizes its own event handling for the model components. Listing 1 gives the variables declaration block along with the initialization procedure of all components (i message).

```
variables multipdevs-simulator
  parent
  tl
  tn
  multiPDEVS = (X,Y,D,{$M_d$})
  with components $M_d$ = ($S_d$,$I_d$,$E_d$,$\delta_{ext,d}$
      ,$\delta_{int,d}$,$\delta_{con,d}$,$\delta_{reac,d}$,$\lambda_d$,$ta_d$)
    with states $q_d$ = ($s_d$,$e_d$)
    a bag of states $k_d^b$
    time of last event $tl_d$ and time
        of next event $tn_d$
    and local outputs $y_d^b$
  event-list = list of components
      $d \in D$ sorted by $tn_d$
end variables

when receive i-message (i, $t$) at
    time $t$
  for each component $d \in D$
    $tl_d \leftarrow t - e_d$
    $tn_d \leftarrow tl_d + ta_d((...,q_i,...))$
  end for
  for each component $d \in D$
    sort $d$ into event-list using $tn_d$
  end for
  $tl \leftarrow$ max{$tl_d$ | $d \in D$}
  $tn \leftarrow$ min{$tn_d$ | $d \in D$}
end when
```

Listing 1: Variable declaration and initialization procedure of the multiPDEVS abstract simulator.

As shown in Listing 1, we keep track of the time of last event tl and time of next event tn for the multiPDEVS but also for all its components (tn_d and tl_d). Upon receipt of an initialization message (i,t), last and next event times tl_d and tn_d of each component are set before sorting them into the event list. Then, the global tl and tn of the multiPDEVS are respectively set to the maximum tl_d and to the minimum tn_d.

Since multiPDEVS allows for external and output events, we give the procedures associated with the corresponding messages in Listing 2. x-messages come from the parent coordinator and carries inputs for the multiPDEVS, and @-messages also come from the parent coordinator to collect outputs of the multiPDEVS.

```
when receive x-message (x, $t$) at
    time $t$ with input value $x_e^b$
  add sub-bag $x_e^b$ to the bag $x^b$
end when

when receive @-message (@, $t$) at
    time $t$
  if $t \neq tn$ then
    error: bad synchronization
  end if
  for each imminent component $d$
      with $tn_d = tn$ and defined $\lambda_d$
    $y_d^b \leftarrow \lambda_d(\ldots,(s_i,t-tl_i),\ldots)$
    add sub-bag $y_d^b$ to the output bag
        $y^b$
  end for
  send (y, $t$) to parent coordinator
      with output bag $y^b$
end when
```

Listing 2: Output (@-message) and input (x-message) procedures of the multiPDEVS abstract simulator.

Upon receipt of an external input message (x,t), the simulator does nothing more than filling the input bag, as a PDEVS simulator does. Upon arrival of a collect message (@,t), the simulator collects all outputs from components that defined the output function λ_d and sends them back to the parent coordinator.

Finally, the procedure associated with the receipt of an internal message (*) is given in Listing 3.

```
40  when receive *-message $(*, t)$ at
      time $t$
    if not $(tl \leq t \leq tn)$ then
      error: bad synchronization
    end if

    if $t = tn$ and $x^b = \varnothing$ then
      for each imminent component $d_\sharp$
          with $tn_{d_\sharp} = tn$
        ps $\leftarrow \delta_{int,d_\sharp}(\ldots,(s_i, t - tl_i),\ldots)$
        for each $s_j$ in ps where $j \in E_{d_\sharp}$
          add $(s_j, d_\sharp)$ to the bag of
              suggested states $k^b_j$
50      end for
      end for
    else if $x^b \neq \varnothing$ then
      for each component $d \in D$
        if $t = tn$ and $tn_d = tn$ then
          if defined $\delta_{ext,d}$ then
            ps $\leftarrow \delta_{con,d}((\ldots,(s_i, t - tl_i),\ldots), x^b)$
          else
            ps $\leftarrow \delta_{int,d}(\ldots,(s_i, t - tl_i),\ldots)$
          end if
60      else
          if defined $\delta_{ext,d}$ then
            ps $\leftarrow \delta_{ext,d}((\ldots,(s_i, t - tl_i),\ldots), x^b)$
          end if
        end if
        for each $s_j$ in ps where $j \in E_d$
          add $(s_j, d)$ to the bag of
              suggested states $k^b_j$
        end for
      end for
    end if
70
    for each $k^b_j$ where $j \in D$ and $k^b_j \neq \varnothing$
      $q_j \leftarrow (\delta_{reac,j}(k^b_j, (s_j, t - tl_j)), 0)$
      $tl_j \leftarrow t$
    end for
    for each $j \in D$ where $k^b_j \neq \varnothing$
      $tn_j \leftarrow tl_j + ta_j(\ldots,(s_i, t - tl_i),\ldots)$
      reschedule $j$ into event-list
    end for

80  $tl \leftarrow t$
    $tn \leftarrow$ min{ $tn_d$ | $d \in D$ }
  end when
```

Listing 3: Internal transition handling procedure of the multiPDEVS abstract simulator.

The procedure associated with the receipt of an internal event $(*, t)$ consist in three sequential steps (Listing 3). The first step (lines 45–69) activates appropriate state transitions for components of the multiPDEVS depending on their tn_d, if there is incoming input from the overall system and if $\delta_{ext,d}$ is defined. When the input bag x^b is empty, we employ the event list and retrieve the imminent components with $tn_{d_\sharp} = t$.

The state transition function $\delta_{int,d_\sharp}(\ldots,(s_i, t - tl_i),\ldots)$ of each imminent $d_\sharp$ is executed (where $(\ldots,(s_i, t - tl_i),\ldots)$ is the state vector of the influencing components $i \in I_{d_\sharp}$) and produce a set of proposed states (ps) for all influenced components $j \in E_{d_\sharp}$. The incoming bag of proposed states is filled by adding to each suggested state the identity of its producer.

When the input bag x^b is not empty and that $t = tn$, the external transition function $\delta_{ext,d}$ is applied for all components $d \in D$ with $tn_d > tn$ that defined $\delta_{ext,d}$. For each imminent components ($tn_d = tn = t$), we activate the confluent transition function $\delta_{con,d}$ if the external transition function $\delta_{ext,d}$ is defined, otherwise we activate the internal transition function $\delta_{int,d}$. Note that all three state transition functions all produce an outgoing set of proposed states for all its influenced components which is then translated in order to fill their incoming bag.

The second step (lines 71–74) consist in activating the reaction transition function for all components having a non-empty bag of incoming proposed states generated by imminent components during the first step. Each component being influenced will produce a new state given all proposed states and its current total state.

The third step (lines 75-81) role is to update the time of next events for each influenced component. As a result of the second step, which update states of influenced components, a chance is given to those components to update their time advance values. This requires the tn_j to be updated as well as a rescheduling of the component in the event list. Finally, the global event times tl and tn are updated appropriately.

Note that the abstract simulator we present in this section is fully sequential. Another version that parallelize processing of components during each micro-step of the *-message procedure with a sync barrier between the two steps can be considered. However, locks associated with influencers states of a given component should be owned by this component each time it is susceptible of reading them. This is a potential source of deadlocks that we do not explore in this paper. Similarly, a distributed version may be considered. Same comments apply with an additional constraint: a mechanism should be provided to allow a component to access its influencers states that may be located on a different node (*e.g.* through the proxy design pattern).

In discrete event simulation, the dynamics of a model can be represented by different "world views", namely the event scheduling world view, the activity scanning world view, the three phase approach world view and the process interaction world view. The abstract simulator we present here belongs to the first approach, where components preschedule their time of execution in the future via the ta_d function. Zeigler et al. (2000) do provide two abstract simulators for the original multiDEVS formalism, one for the event scheduling world view, and another both for the activity scanning and the process interaction world views. For multiPDEVS, those alternative abstract simulators can be considered for further research but we do not present them in this paper.

This last subsection, which defines the abstract simulator, provides behavioral semantics to the structural definition of multiPDEVS given in section 3.1. Added to the proof that a multiPDEVS may exist within a larger PDEVS simulation (cf. section 3.2), all necessary information are given in order to realize a proper implementation of the multiPDEVS formalism.

4 Case study : application to fire spread modelling

Based on a fire spreading model (Balbi et al., 1999), this section propose to illustrate the modeling process using a multicomponent approach. First subsection introduces the semi-physical fire spread model. Then, second subsection presents the related multiPDEVS specification. Finally, last subsection gives comparison with a modular formalism, namely Cell-DEVS (Wainer and Giambiasi, 2001).

4.1 Semi-physical fire spread model

Among the variety of mathematical models that grasps fire propagation, we focus on a *semi-physical* model (Balbi et al., 1999). According to Weber (1991), fire spread models can be classified in three categories depending on their properties : *statistical* models, which do not include any physical phenomena; *semi-empirical* models, which consider the principle of energy conservation and finally, *physical* models, being the most detailed models. The model we use is a combination of the last two categories, and is described as a non-stationary two-dimensional *semi-physical* model (Balbi et al., 1999). It is a semi-empirical model transposed to integrate two dimensions and which consider physical properties such as heat transfer for propagation.

The model is spatialized through elementary cells holding plant mass, where each cell is described by the following partial differential equation:

$$\frac{\delta T}{\delta t} = -k(T - T_a) + K\Delta T - Q\frac{\delta \sigma_v}{\delta t} \tag{1}$$

where

$$\sigma_v = \begin{cases} \sigma_{v0} & \text{if } T < T_{ig} \\ \sigma_{v0}e^{-\alpha(t-t_{ig})} & \text{if } T \geq T_{ig} \end{cases} \tag{2}$$

and

$$T_{x,y} = T_a \quad \text{at the boundary} \tag{3}$$
$$T_{x,y} \geq T_{ig} \quad \text{for the burning cells} \tag{4}$$
$$T_{x,y} = T_a \quad \text{for the non-burning cells at } t = 0 \tag{5}$$

Parameter		Description
T_a	(27 °C)	ambient temperature
T_{ig}	(300 °C)	ignition temperature
T	(°C)	temperature
K	($\text{m}^2\,\text{s}^{-1}$)	thermal diffusivity
Q	($\text{m}^2\,°\text{C}\,\text{kg}^{-1}$)	reduced combustion enthalpy
Δ		Laplacian operator in two-dimensional Cartesian coordinates
α	(s^{-1})	combustion time constant
σ_v	($\text{kg}\,\text{m}^{-2}$)	vegetable surface mass
σ_{v0}	($\text{kg}\,\text{m}^{-2}$)	initial vegetable surface mass (before combustion)
t_{ig}	(s)	ignition time

Table 1: Nomenclature of fire spread model parameters (Balbi et al., 1999).

Table 1 gives the nomenclature of model parameters. The model parameters are identified by Balbi et al. (1999) from experimental data of temperature versus time.

This particular model has been discretized in Muzy et al. (2005) using the Finite Difference Method (FDM), which leads to the following algebraic equation:

$$T_{i,j}^{k+1} = a(T_{i-1,j}^{k} + T_{i+1,j}^{k}) + b(T_{i,j-1}^{k} + T_{i,j+1}^{k}) + cQ(\frac{\delta \sigma_v}{\delta t})_{i,j}^{k} + dT_{i,j}^{k} \quad (6)$$

where $T_{i,j}$ represent the cell temperature, and where a, b, c and d are coefficients that depends on the time step and mesh size considered. For our example, as in Muzy et al. (2005), we consider a discrete time step of 0.01 s and uniform cells of 1 cm^2.

Such discretized model is well and easily expressed using a multicomponent approach since a cell can be represented by a component and its neighbors can be represented using the set of influencing components I_d and the set of influenced components E_d. To illustrate how such system can be modelled using multiPDEVS, we propose a detailed specification in the next sub-section.

4.2 multiPDEVS specification

In this subsection, we propose a specification of the semi-physical fire spread model using multiPDEVS to represent a cellular automaton, where each cell holds its temperature. When the automaton evolves, each cell updates its temperature according to ones of its Moore neighborhood. The multiPDEVS model represents the whole surface, which is composed of homogeneous components with identical behavior influencing each other uniformly.

The multiPDEVS can be specified as follows:

$$multiPDEVS = (X, Y, D, \{M_{x,y}\})$$

X and Y are empty sets since there is no input or output to the model. $D = \{(x,y) \mid x \in I, y \in I\}$ is the index set composed of two-dimensional coordinates. For each $(x,y) \in D$, the component, which represents a cell, is specified as:

$$M_{x,y} = (S_{x,y}, I_{x,y}, E_{x,y}, \delta_{int,x,y}, \delta_{reac,x,y}, ta_{x,y})$$

The state $S_{x,y}$ of a component cell is represented by the following quintuple:

$$S_{x,y} = \{(phase, T, T_{prev}, T_{neigh}, t_{ig})\}$$

where $phase \in \{inactive, warming, burning\}$, $T \in \mathbb{R}_0^+$ is the current temperature of the cell, $T_{prev} \in \mathbb{R}_0^+$ is the previous temperature of the cell, $T_{neigh} = (\dots, T_i, \dots)$ with $i \in E_i$ is the temperature vector of all neighbors and finally, $t_{ig} \in \mathbb{R}_0^+$ holds the simulation time at which the cell is ignited.

The set of influencers is defined by the Moore neighborhood and the cell itself: $I_{x,y} = \{\ (x,y), (x,y+1), (x+1,y), (x,y-1), (x-1,y), (x+1,y+1), (x+1,y-1), (x-1,y+1), (x-1,y-1)\}$. Reciprocally, a cell is able to influence itself and its neighbors. Thus, the set of influencees $E_{x,y} = I_{x,y}$.

$\delta_{ext,x,y}$, $\delta_{con,x,y}$ or $\lambda_{x,y}$ are not specified since there is no overall input or overall output.

Basically, for a given cell (x,y), when $T_{x,y} - T_{prev,x,y}$ reaches a threshold, (x,y) updates for all its influencees $(i,j) \in E_{x,y}$ its corresponding temperature in the temperature vector $T_{neigh,(i,j)}$ with its current temperature $T_{x,y}$. At each step, the cell (x,y) also produces a new state for itself ($(x,y) \in E_d$). It means that each cell may be given nine potential states to its $\delta_{reac,x,y}$ function (eight proposed states from its neighbors plus one suggested for itself). In this particular case, the reaction transition function composes a new state using primarily the proposed one produced by itself, but constructs the T_{neigh} temperature vector using proposed states from other components.

Note that in order to simplify transitions functions specification of the fire spread model, we will use the "dot-notation", conventionally used in object-oriented programming languages (*e.g.* the element $q_x.T$ must be understood as the temperature element of the global state q_x of component x).

Since each cell holds a vector of nearby temperatures that is updated by neighbors themselves, it is possible to keep some cells inactive, that is inactive cells ($phase = inactive$, $T_{x,y} = T_a$). This results in the following time advance function:

$$ta_{x,y}(s) = \begin{cases} \infty & \text{if } s.phase = inactive \\ 0.01 & \text{otherwise} \end{cases}$$

Given that the definition of $\delta_{int,x,y}$ and $\delta_{reac,x,y}$ are much less concise to write, we specify these functions in the following pseudo-code:

```
variables:
    $a$, $c$, $d$, $\alpha$ // equation coefficients
    $t$
function $\delta_{reac,x,y}(k^b_{x,y}, s_{x,y}, e_{x,y})$ do
    $s'_{x,y} \leftarrow s_{x,y}$
    $T'_{neigh} \leftarrow s_{x,y}.T_{neigh}$
    $\forall (s,(i,j)) \in k^b_{x,y}$ do
        if $(i,j) = (x,y)$ then // self influence
            $s'_{x,y} \leftarrow s$
        else
            $T'_{neigh}.T_{i,j} \leftarrow s.T_{neigh}.T_{i,j}$
            if $s_{x,y}.phase = inactive$ then
                $s'_{x,y}.phase \leftarrow warming$
```

```
          end if
        end if
      end
      $s'_{x,y} . T_{neigh} \leftarrow T'_{neigh}$
      return $s'_{x,y}$
    end function

20  function $\delta_{int,x,y}((s_{x,y}, e_{x,y}), \dots, (s_{i,j}, e_{i,j}), \dots)$ do
      sum $\leftarrow \sum_{(i,j) \in I_{x,y} \setminus \{(x,y)\}} s_{x,y} . T_{neigh} . T_{i,j}$
      $s'_{x,y} \leftarrow s_{x,y}$
      $s'_{x,y} . T_{prev} \leftarrow s_{x,y} . T$
      $\forall (i,j) \in I_{x,y} \setminus \{(x,y)\}$ do
        $s'_{i,j} \leftarrow s_{i,j}$
        $s'_{i,j} . T_{neigh} . T_{x,y} \leftarrow s_{x,y} . T$
      end
      if $s_{x,y} . phase = warming$
        $s'_{x,y} . T \leftarrow d * T + a * sum + c$
30    else if $s_{x,y} . phase = burning$
        $s'_{x,y} . T \leftarrow d * T + a * sum + c * \exp(-\alpha *$
            $(t - t_{ig}))$
      end if
      $s'_{x,y} . phase \leftarrow newphase(s_{x,y} . phase, s'_{x,y} . T)$
      if $s_{x,y} . phase = warming \wedge s'_{x,y} . phase = burning$
        $s'_{x,y} . t_{ig} \leftarrow t$
      end if
      return $(s'_{x,y}, \dots, s'_{i,j}, \dots)$
    end function

40  function newphase($phase$, $T$) do
      if $T \geq T_{ig}$ then
        return $burning$
      else if $T > T_a$
        return $warming$
      end if
    end function
```

Listing 4: Specification of $\delta_{int,x,y}$ and $\delta_{reac,x,y}$ functions in pseudo-code for the multiPDEVS fire spread model example.

The $\delta_{reac,x,y}$ function defined in Listing 4 allows a (x,y) cell to decide its new state given its current state $q_{x,y}$ and all suggested ones in $k^b_{x,y}$. When iterating over suggested states (line 6), if a suggested state was produced by (x,y) for itself through the $\delta_{int,x,y}$ function ($(i,j) = (x,y)$, line 7), the next state is primarily constructed based on this suggested state. If there is no suggested state produced by (x,y) (cell is inactive), the state is primarily based on the current one (line 4). For other producers of suggested states ($(i,j) \neq (x,y)$), the cell (x,y) is only interested in the temperature vector that was updated by the producer, so it constructs its new temperature vector according to all these contributions (line 10). If the cell is currently inactive while receiving nearby temperatures, the cell will pass in the warming phase (line 11–13).

The $\delta_{int,x,y}$ function suggests new states for all its influencees. Two parts can be identified. The first (lines 21–27) corresponds to the construction of new states for neighbors. Those states are simply a copy of their current states with an update to the temperature vector so that the current temperature corresponding to $((x,y))$ is updated. The second phase (lines 28–36) corresponds to the calculation of the new temperature for this cell, according to equations given in subsection 4.1. The conditional statement allows to compute the σ_v value depending on the macro state of the cell, according to (2). Regarding (6), coefficients a and b correspondong to the thermal conductivity of neighbors cells are merged into a since we use the same value. This coefficient is applied to the sum of neighbors temperatures, which is computed line 21. A new phase is then calculated using the *newphase* function depending on the new temperature and the current phase. Finally, if the cell phase pass from warming to burning, the ignition time is set.

We should emphasize that we modelled component cells this way as a proof of concept of state collisions handling, but such a model could be modelled the other way around using only the set of influencers $I_{x,y}$ to read neighbors states temperatures and restrict $E_{x,y}$ to the unit set $\{(x,y)\}$ so that each cell generate a state only for itself. However, this alternative approach would require to change the time advance functions in order to keep all cells active so that they are able to poll neighbors temperatures even if they are inactive. In practice, the former approach prevent inactive cells to be activated until they are warmed up by neighbors while the latter approach keeps all cells active.

4.3 Comparison with a modular formalism

We discuss here the benefits of specifying a system such as the fire spread model described earlier using multiPDEVS in comparison with a modular formalism such as PDEVS or Cell-DEVS.

From a modelling perspective, the same fire spread model specified through a network of components such as a PDEVS coupled model instead of a multiPDEVS is much more verbose to specify because all couplings between cells have to be addressed since cells are represented by atomic models. The Cell-DEVS (Wainer and Giambiasi, 2001) extension, on the other hand, ease the specification of cellular automata by abstracting the definition of couplings between neighbor cells while preserving modularity, these last still being atomic models. As of multiPDEVS, it is well suited for the specification of cellular automata as-is for two reasons: neighborhood is easily represented through the influencer/influencee principle; and components are able to access their neighbors states without having to anticipate variables of interest and having to send these through dedicated functions as this is the case for PDEVS or Cell-DEVS.

From a simulation perspective, using a modular approach may be less advantageous depending on the implementation since event routing is a potential source of overhead (Muzy and Nutaro, 2005), although several techniques (Chen and Vangheluwe, 2010; Himmelspach and Uhrmacher, 2006) allow to manage this issue. In contrast, multiPDEVS has no communication overhead between components influencing each other since it avoids traditional event routing.

We realized an implementation of multiPDEVS using Quartz[1], a Crystal port of DEVS-Ruby (Franceschini et al., 2014), which is a simulation tool that allows specification of PDEVS models. It provides several extensions such as Cell-DEVS (Wainer and Giambiasi, 2001) or DSDE (Barros, 1995), and we extended it to integrate multiPDEVS and its simulator. To define new multiPDEVS models, Quartz provides a class MultiComponent::Model which can be modularly coupled to other models and to which may be added component models. Component models can be defined using the provided abstract class MultiComponent ::Component that the modeler has to extend via inheritance. Our tool then use the appropriate simulator according to the model type during simulation.

Figure 5 illustrates execution of the firespread model using multiPDEVS (as given in section 4.2) on a 25x25 grid at different simulation times. Regarding parameters values, we used the constant temperature values described in Table 1. Also, the reduced combustion enthalpy Q was set to 2.74, and the combustion time constant α was set to 0.19. Coefficients a and b from (6), which corresponds to the thermal conductivity of neighbors cells were both set to 0.0031. Coefficient d, which corresponds to the thermal conductivity of the actual cell was set to 0.98689. Finally, coefficient c was set to 0.213.

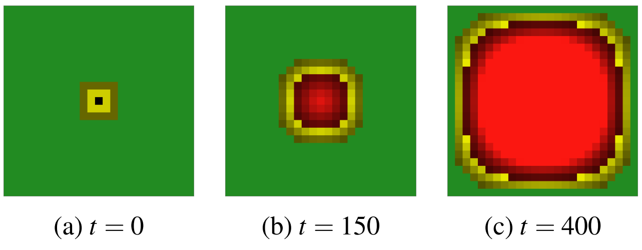
(a) $t = 0$ (b) $t = 150$ (c) $t = 400$

Figure 5: Execution of a simple fire spreading model on a 25x25 grid of cells at times 0 (5a), 150 (5b) and 400 (5c).

[1]Code available at https://github.com/rumenzu/quartz.

An initial hotbed exists at the center of the grid where one cell is ignited and surrounding cells are warmed (Fig. 5a). Temperature in the rest of the cell space is homogeneous and represent the ambient temperature. This simple model allows us to test our implementation using highly interacting low computational components that remains active during the whole simulation.

In order to verify our intuition that we may obtain best execution times using multiPDEVS due to the absence of event routing, we implemented the same model using the Cell-DEVS extension of Quartz in order to compare both results. For this performance analysis, we ran ten simulations for each approach (multiPDEVS and Cell-DEVS) using the same model, the same initial conditions and the same environment (Quartz) and measured the elapsed real time. The test environment is based on a Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz (3MB L2 cache), 8GB (2x DDR3L - 1600Mhz) of RAM, an Apple SSD SM128E hard drive, running on OSX 10.11.4. Software used is Quartz. Figure 6 shows the results of running those simulations with error bars showing average, min and max measured times for each batch of repeated simulations and Table 2 shows average elapsed real time results and the relative standard deviation for each experiment.

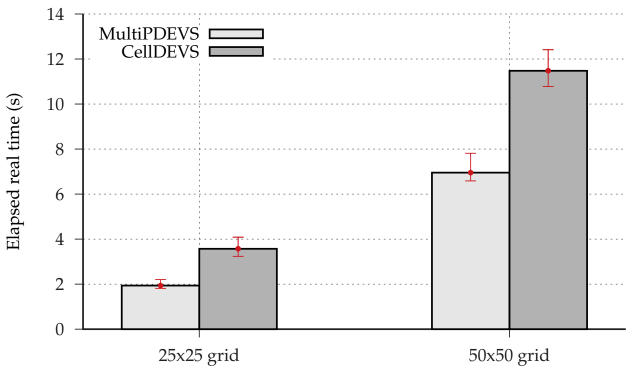

Figure 6: Performance comparison of multiPDEVS and Cell-DEVS showing elapsed real time in seconds of a fire spread simulation for a grid of 25x25 and a grid of 50x50 with a fixed simulation duration of 1200 and across 10 runs.

Event routing and message passing overhead in Quartz is reduced using techniques similar to Himmelspach and Uhrmacher (2006) and Vicino et al. (2015). Despite that, as we can appreciate on the

Approach	Grid	Avg. ela. time (s)	Rel. std. dev.
Cell-DEVS	25x25	3.5707911	± 6.43%
	50x50	11.4745384	± 3.54%
MultiPDEVS	25x25	1.9359938	± 6.69%
	50x50	6.9516354	± 6.59%

Table 2: Performance comparison results of multiPDEVS and Cell-DEVS showing average elapsed real time and relative standard deviation for each approach.

graph, multiPDEVS yields better results than its modular counterpart for the same simulated model as we obtain an average speedup of 1.75x. Those results comfort us in multiPDEVS relevance, especially for these kind of highly communicative models with tight coupling. However, in order to extract better conclusions relative to performances, a new benchmark is necessary using a framework that performs complete flattening (as mentioned in section 2), *ie* transformation of the Cell-DEVS in a coupled model of depth one (which Quartz does) and the transformation of this coupled to an atomic model.

In this section we presented a fire spreading model cellular automaton. Cellular automata is a modelling paradigm that is well expressed using the multiPDEVS formalism and it was also the opportunity to compare its performances against a Cell-DEVS implementation. The next section discuss benefits and drawbacks of the proposed approach from modelling and simulation perspectives.

5 Discussion

The fire spread example shows that multiPDEVS can be used to fully specify a system. However, it is quite legitimate for the modeler to ask the following question: ”When should I, or should I not use this formalism ?”. MultiPDEVS can improve the modelling process in a multi-formalism context, where modular and non-modular system specification are used at the same time. We believe that the modular approach proposed by PDEVS, and the non-modular one have to be seen as complementary approaches, and in no way as competing approaches. Regularly, systems offers a clear hierarchical description coupled with elements where the system behavior will be harder to describe using modular specification formalism. In those cases, it can be more comfortable to describe the overall interaction through direct influence mechanism rather than describe the system at a higher level of specification. The modular approach will allow to bring a good intelligibility of the model, while the non-modular multicomponent approach will allow to simplify the description of some phenomenons by improving the formalism expressiveness.

One can ask the question of the reusability of a non-modular model, and more particularly of these various components. If with the fire spread example, such question does not really arise (components are the same), it becomes legitimate for models where components differ. By nature, the influencer/influencee principle used with non-modular approach makes components strongly dependants. Obviously it is possible to reuse or replace components of a multicomponent model, however, this must be done with caution. The modeler needs to perfectly understand interactions with influencee components and influenced components of the targeted component. Without making reusability as a trivial process, the modular approach will help since more effort was previously given to express interactions between components. Reusability at the multicomponent level is equivalent to the modular approach, since a multiPDEVS model is strictly equivalent to a PDEVS atomic model.

As defined in 3.3, a multiPDEVS model can be integrated within a PDEVS-based simulation environment using its own dedicated simulator and can be simulated through an abstract mechanism, as originally defined in Zeigler et al. (2000). The whole simulation is driven by several *processors* (*cf.* Figure 4), where *coordinators* are responsible for managing event routing and scheduling their children and *simulators* are responsible for the activation of their associated model. The multiPDEVS simulator we provide perfectly conforms to this mechanism, and thus, allows multiPDEVS to be modularly coupled with other PDEVS models and executed by the same PDEVS coordinator. This allows to reduce the difficulty of integrating multiPDEVS within a simulation environment while enriching the set of formalisms available for the user of such platform. Due to the multiPDEVS simulation protocol nature, we observe traffic messages reduction in relation to PDEVS. Such observations suggests increased simulation performances. Since an appropriate benchmark has not been done yet, and that some work (Muzy and Nutaro, 2005) shows that proper management of messages can significantly increase performances of PDEVS, we will remain cautious in the final performance of multiPDEVS.

Regarding performances, multiPDEVS may be affected by the nature of interactions between components. In the fire spread example we used, components states have small memory requirements, but we can ask how well the formalism scales with larger memory needs. Since the sets of suggested states for each components are temporarily kept during each simulation cycle, memory allocation can be a concern. We should note that PDEVS has a similar issue at a lower extent with input messages, instead of suggested states. For example, with PDEVS, messages sent between two components can be sized according to the needs of the model. Contrariwise, to change a single value of the state of an influencee, a multiPDEVS component should construct a whole new state for its influencee. In practice, pointers techniques could help reduce this overhead, but this particular issue should be further studied to have a better idea of multiPDEVS performances.

Another element of performance improvement may appear with some types of models. If we focus in communications between components, a multiPDEVS may require fewer simulation cycles than PDEVS in some cases. Lets take two components, A and B, where B needs informations provided by A. Here is PDEVS and multiPDEVS way to modelize it: Using PDEVS: B can ask A for data. Then A send data to B (see figure 7). Following PDEVS simulation protocol, this will therefore require at least two simulation cycles. Using multiPDEVS: B can directly read data from A (A is an influencer of B). Following the multiPDEVS protocol, a single simulation cycle will therefore be necessary. Note that such example consider only the case where B is explicitly requesting informations known by A component. We intentionally forget cases where A is a data generator, of where A needs computation time to provide data. An alternative to multiPDEVS and PDEVS is HFSS (Barros, 2003), which allows components to sample inputs from their influencers through couplings in a one-step process, similarly to multiPDEVS. Since HFSS preserves modularity through couplings, it lies at a higher level of specification than multiPDEVS. However, multiPDEVS is pertinent for cases where component states need to be directly accessible.

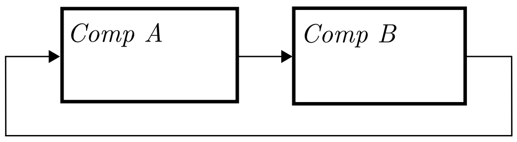

Figure 7: Communication between two components

Further studies would be needed to target the classes of problems for which this type of property would be an added value. We are thinking for the moment of the domains where agent and cellular automata are used jointly (e.g. Agent interrogating its environment modeled by several cells).

Regarding some design choices about multiPDEVS, we decided to stay as close as possible to the original multiDEVS. However, an interesting alternative regarding multiPDEVS semantics may be considered. As multiPDEVS is defined, only components that have been influenced (*i.e.* that updated their state via their reaction function δ_{reac}) have a chance to update their time of next event. Since the time advance function of a component depends on all influencers states, we could also give the opportunity to all components having at least one influencer with a new state to update its tn. This would open new modeling perspectives. As for example, the case study we propose in section 4.2, could be simplified. Basically, an inactive cell could decide if it is the appropriate moment to enter a burning state each time one of its neighbors is updated.

Generally speaking, DEVS-based formalisms where components are allowed to communicate at the same simulation time, such as PDEVS or multiPDEVS, do not offer direct mechanisms to confirm to other components the use of an information/resource which was provided to them. This can raise issues of inconsistency unique to such formalisms. To illustrate this, imagine particles that can move from one cell to a neighbor as shown Figure 8.

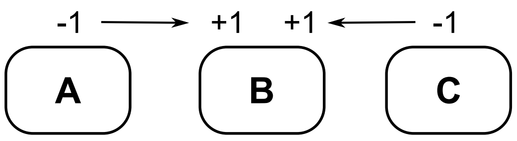

Figure 8: Particles example: two particles proposed to cell B.

The state of the model is in $\{0,1\}$. Suppose particles in the A and C cells want to move in to the center cell B at the same time. A and C send their proposed states. Since decision is based on proposed states, the center can choose a +1 (if it empty) but that leaves open which neighbor is the sender. If it chooses one (by some rules), it must inform the neighbors so they will adjust their states accordingly. Currently, this feedback to maintain consistency will require a second complete simulation phase. Now, multiPDEVS introduced the K set (whose elements are tuples composed of suggested states and of the identity of the component that sent it), specifically for this purpose, so that a component can choose its new state depending on the identity of the sender (which is not necessarily present in its set of influencers I). In the particle example described using multiPDEVS, each component would be responsible to adjust its neighbours' states to ensure that no particle is lost in the system. This also means that a state described by $S = \{0,1\}$ is not sufficient, since the component has to store information about particles to be sent back through the classical δ_{int} function and a $ta(s) = 0$. At this moment, it seems that such an issue is best tackled using a modelling approach (which involves use of ta(s)=0 loops) rather than expand the formalism itself. Nevertheless, an extension of multiPDEVS formalism might be to allow some kind of built in reverse propagation of selection information and this can be considered as a source of further research.

6 Conclusion

TMS provides a DEVS specialisation dedicated to non-modular modelling, named multiDEVS. As DEVS, multiDEVS formalism comes with a lack of expressiveness to properly manage collisions between events. PDEVS formalism has been proposed to fill this gap for the modular approach. Hence, we propose multiPDEVS, who furnish an effective management of events conflicts as proposed in PDEVS, to multiDEVS. The multiPDEVS formalism supports concepts as introduced by PDEVS, such as the bag concept to combine simultaneous event into a single one, and the confluent transition function to combine internal and external transition function when they occur at the same simulation time. In contrast to PDEVS, multiPDEVS handles an additional kind of conflict dictated by the non-modular approach, which we call *state collisions*. Such *state collisions* appear when multiple components execute state transitions function at the same simulation time, resulting in a possible violation of other components autonomy. We are convinced that autonomy of components is a necessary property for a proper modeling process. To keep such property, multiPDEVS provides new concepts around δ_{reac} function. These new concepts give modelers the possibility to collect *state collisions* in a bag of states (K^b) and then manage them explicitly at component level through the reaction function (δ_{reac}). The multiPDEVS formalism offers to facilitate the modeling process for non-modular approaches without increasing message exchanges. The multiPDEVS formalism is proved to be equivalent to a well-defined atomic PDEVS model. Such property involve the possibility to integrate multiPDEVS in a larger PDEVS simulation.

The multiPDEVS formalism has been compared to others non-modular formalisms such as CellSpace. We show that CellSpace can be considered as a restriction of multiPDEVS. Finally, the multiPDEVS formalism has been implemented using Quartz and allowed us to test its performances against a Cell-DEVS approach using a cellular automaton. This performance speedup added to good modelling abilities comforted us in its significance for highly communicative models with tight coupling but also for modelling paradigms falling under bottom-up approaches.

Definition of the multiPDEVS formalism opens many perspectives, from both an utility and an extensibility point of view. Future works includes use of the multiPDEVS formalism to describe MAS environments as cellular models since multiPDEVS is well-suited to represent spatially explicit models. Individual-based models (IBM) are also good candidates to be defined using multiPDEVS, where individuals are represented with components interacting with each other. Besides, we predict a simulation speedup using multiPDEVS rather than PDEVS both for representing MAS environments and IBM for the same reason we obtained better results representing a cellular automaton using multiPDEVS, which is the reduction of message exchange between models during simulation.

Finally, the 5 attempts to dissociate the simulation aspect from the modelling aspect to provide a clear vision of pros and cons of using multiPDEVS formalism.

Currently in full-scale testing in a fisheries management project, the multiPDEVS formalism should allow efficient modeling of fishing areas and interactions between them. As for the future of the multiPDEVS formalism we consider many perspectives of evolution. We plan to define alternative abstract simulators following other simulation strategies, known as world views (*cf.* section 3.3). We also consider as a further research the definition of a parallel and distributed version of the abstract simulator. Another interesting property applied to multiPDEVS would be that of dynamic structure, which DEVS and PDEVS benefits from via DS-DEVS (Barros, 1995) or dynDEVS (Uhrmacher, 2001) extensions. For completeness, we also consider definition of a multicomponent parallel discrete time system, multiPDTSS, along with its abstract simulator. Given that discrete time is a special case of discrete event systems, multiPDTSS would allow to explicitly combine discrete time systems with discrete event systems within the same framework.

As mentioned previously, there are other possible approaches, especially HFSS (Barros, 2003). It would be very interesting to make a deeper comparison with it, especially on the ergonomic aspect of modeling and performance.

Acknowledgments

This work is carried out within the framework of the MoonFish research project supported by the Corsican region (CTC) and the European Union (EU) through the po-FEDER regional programs. The authors declares that there is no conflict of interest regarding the publication of this paper.

References

Al-Habashna, A. and G. Wainer (2016). "Modeling pedestrian behavior with Cell-DEVS: theory and applications". In: *Simulation* 92.2, pp. 117–139 (cit. on p. 2).

Bae, J. W. et al. (2016). "Efficient Flattening Algorithm for Hierarchical and Dynamic Structure Discrete Event Models". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 26.4, p. 25 (cit. on pp. 2, 3).

Balbi, J., P. Santoni, and J. Dupuy (1999). "Dynamic modelling of fire spread across a fuel bed". In: *International Journal of Wildland Fire* 9.4, pp. 275–284 (cit. on p. 13).

Barros, F. J. (1995). "Dynamic structure discrete event system specification: a new formalism for dynamic structure modeling and simulation". In: *Proceedings of the 27th conference on Winter simulation*. IEEE Computer Society, pp. 781–785 (cit. on pp. 16, 19).

– (2003). "Dynamic Structure Multiparadigm Modeling and Simulation". In: *ACM Transactions on Modeling and Computer Simulation* 13.3, pp. 259–275 (cit. on pp. 18, 19).

Chen, B. and H. Vangheluwe (2010). "Symbolic flattening of DEVS models". In: *Proceedings of the 2010 Summer Computer Simulation Conference*. Society for Computer Simulation International, pp. 209–218 (cit. on pp. 2, 16).

Cho, S. M. and T. G. Kim (1998). "Real-time DEVS simulation: Concurrent, time-selective execution of combined RT-DEVS model and interactive environment". In: *Koasas* (cit. on p. 1).

Chow, A. C. H. and B. P. Zeigler (1994). "Parallel DEVS: a parallel, hierarchical, modular, modeling formalism". In: *Proceedings of the 26th conference on Winter simulation*. Society for Computer Simulation International, pp. 716–722 (cit. on pp. 1, 4).

Chow, A. C. H., B. P. Zeigler, and D. H. Kim (1994). "Abstract simulator for the parallel DEVS formalism". In: *Proceedings of the Fifth Annual Conference on AI, Simulation, and Planning in High Autonomy Systems, 1994. Distributed Interactive Simulation Environments*. IEEE, pp. 157–163 (cit. on p. 11).

Franceschini, R. et al. (2014). "DEVS-Ruby: a Domain Specific Language for DEVS Modeling and Simulation (WIP)". In: *DEVS 14: Proceedings of the Symposium on Theory of M&S*. SCS International, pp. 393–398 (cit. on p. 16).

Himmelspach, J. and A. M. Uhrmacher (2006). "Sequential Processing of PDEVS models". In: *Proceedings of the 3rd EMSS*, pp. 239–244 (cit. on p. 16).

Hwang, M. H. and B. P. Zeigler (2009). "Reachability graph of finite and deterministic devs networks". In: *IEEE Transactions on Automation Science and Engineering* 6.3, p. 468 (cit. on p. 1).

Innocenti, E. et al. (2004). "Active-DEVS: a computational model for the simulation of forest fire propagation". In: *Systems, Man and Cybernetics, 2004 IEEE International Conference on*. Vol. 2. IEEE, pp. 1857–1863 (cit. on p. 2).

Jafer, S. and G. Wainer (2009). "Flattened conservative parallel simulator for DEVS and Cell-DEVS". In: *Computational Science and Engineering, 2009. CSE'09. International Conference on*. Vol. 1. IEEE, pp. 443–448 (cit. on pp. 2, 3).

Kofman, E. and R. Castro (2006). "STDEVS, A Novel Formalism for Modeling and Simulation of Stochastic Discrete Event Systems". In: *Proceedings of AADECA 2006*, pp. 1–6 (cit. on p. 1).

Muzy, A. and J. Nutaro (2005). "Algorithms for efficient implementations of the DEVS & DSDEVS abstract simulators". In: *2008 12th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT)*. IEEE, pp. 273–279 (cit. on pp. 16, 17).

Muzy, A. et al. (2003). "Optimization of cell spaces simulation for the modeling of fire spreading". In: *Proceedings of the 36th annual symposium on Simulation*. IEEE Computer Society, p. 289 (cit. on pp. 2, 3).

Muzy, A. et al. (2005). "Modelling and simulation of ecological propagation processes: application to fire spread". In: *Environmental Modelling & Software* 20.7, pp. 827–842 (cit. on pp. 13, 14).

Shiginah, F. A. and B. P. Zeigler (2011). "A new cell space DEVS specification: Reviewing the parallel DEVS formalism seeking fast cell space simulations". In: *Simulation Modelling Practice and Theory* 19.5, pp. 1267–1279 (cit. on pp. 2, 5).

Shiginah, F. A. (2006). "Multi-Layer Cellular DEVS Formalism for Faster Model Development and Simulation Efficiency”. PhD thesis. The University of Arizona: The University of Arizona (cit. on p. 5).

Uhrmacher, A. M. (2001). “Dynamic Structures in Modeling and Simulation: A Reflective Approach”. In: *ACM Transactions on Modeling and Computer Simulation* 11.2, pp. 206–232 (cit. on p. 19).

Vangheluwe, H. (2008). “Foundations of Modelling and Simulation of Complex Systems”. In: *Electronic Communications of the EASST* 10, pp. 148–162 (cit. on p. 1).

Vicino, D. et al. (2015). “Sequential PDEVS Architecture”. In: *DEVS 15: Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative*. Alexandria, VA, USA, pp. 906–913 (cit. on p. 16).

Wainer, G. A. and R. Castro (2010). “A Survey on the Application of the Cell-DEVS Formalism.” In: *J. Cellular Automata* 5.6, pp. 509–524 (cit. on p. 2).

Wainer, G. A. and N. Giambiasi (2001). “Application of the Cell-DEVS paradigm for cell spaces modelling and simulation”. In: *Simulation* 76.1, pp. 22–39 (cit. on pp. 1, 2, 6, 13, 15, 16).

Weber, R. (1991). “Modelling fire spread through fuel beds”. In: *Progress in Energy and Combustion Science* 17.1, pp. 67 –82. ISSN: 0360-1285. DOI: http://dx.doi.org/10.1016/0360-1285(91)90003-6 (cit. on p. 13).

Zacharewicz, G. et al. (2010). “A generalized discrete event system (g-DEVS) flattened simulation structure: Application to high-level architecture (HLA) compliant simulation of workflow”. In: *Simulation* 86.3, pp. 181–197 (cit. on p. 2).

Zeigler, B. P. and H. S. Sarjoughian (2013). *Guide to Modeling and Simulation of Systems of Systems*. Simulation Foundations, Methods and Applications. London: Springer London. ISBN: 978-0-85729-864-5 978-0-85729-865-2. (Visited on 06/29/2017) (cit. on p. 10).

Zeigler, B. P., H. Praehofer, and T. G. Kim (2000). *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic press (cit. on pp. 1, 2, 6, 13, 17).