Towards a Formal Semantics of Event-Based Multi-agent Simulations

Jean-Pierre Müller

CIRAD, UPR GREEN, Montpellier, F-34398 France
Associate researcher to LIRMM, Montpellier, France

Abstract. The aim of this paper is to define a non-ambiguous operational semantics for event-based multi-agent modeling and simulation, applied to complex systems. A number of features common to most multi-agent systems have been retained: 1) agent proactive as well as reactive behavior, 2) *concurrency*: events can arrive simultaneously to an agent, an environment or any simulated entity and the actual change only depends on the target according to the influence/reaction paradigm [1], 3) *instantaneity*: if reaction takes time, perception as well as information diffusion is instantaneous and should be processed separately, 4) *structure dynamics*: the interaction structure (who is talking to whom) changes over time, and the agents as well as any simulated entity may be created or destroyed in the course of the simulation.

For each of these features, a solution inspired by the work on *DEVS* (Discrete EVent Systems, [2]) is proposed. *Proactive/reactive behavior* is naturally taken into account by *DEVS*. *Concurrency* is dealt with using *//–DEVS* (in [2]), a variant of the pure *DEVS*. *Instantaneity* is managed by distinguishing the physical events producing state transitions and the logical events realizing only perception and information diffusion. The *structure dynamics* is achieved by using a variant of *ρ-DEVS* (cf. [3]) where the expressiveness allows to manage hierarchical structures. The operational semantics is given as abstract algorithms and the expressive power of this formalism is illustrated on a simple example.

1 Introduction

The complex systems are characterized by a set of local components or entities in non-linear interaction whose global behavior is not reducible to any composition of the individual behaviors [4]. The question at the origin of the modeling process largely defines both the system and the entities to consider and therefore the point of view. The problem is even more complicated when it is necessary to articulate a set of these disciplinary points of view, possibly at various levels of organization [5,6], as it is often the case for eco-sociosystems. These disciplinary points of view produce a number of *thematic models* to be articulated. Many *formalisms* have been proposed to model the thematic models either at the aggregated level with differential equations, possibly partial for taking into account the spatiality of the phenomena, the cellular automata or, at the local (non-aggregated) level, individual or multi-agent based systems. An overview of the existing formalisms is given in [6]. Finally the models in these dedicated formalisms are implemented using raw programming languages up to specific simulation *platforms* like MatLab and Stella for dynamical systems, or Cormas [7] and Repast [8] for multi-agent systems, to cite a few. The modeling process going from the thematic model to its expression within a formalism down to the implementation platform needs to be made easier as more and more complex systems have to be modeled.

Our aim is to define an implementation platform such that any formalism can be mapped in a systematic (and then automatizable) way onto it. Among these formalisms, we shall concentrate on multi-agent systems. Theoretically, multi-agent systems are mainly characterized by the openness (entities coming in and out) and structure dynamics (the topology of interaction is dynamical). More than the definition of agents as having purposeful behavior (cf. [9]), we shall retain more generally the following features of the multi-agent based formalisms:

1. *reactive and proactive behavior*: an agents reacts to incoming events (if it wants to), and behaves proactively, regardless of the complexity of its internal architecture (from simple rules up to sophisticated reasoning),
2. *concurrency*: events can arrive simultaneously to an agent, an environment or any simulated entity. We consider that the actual change only depends on the target according to the influence/reaction paradigm [1], and that the result must not depend on the order of arrival nor on the order in which the entities are run as in many existing platforms,
3. *instantaneity*: if reaction takes (simulated) time, perception as well as information diffusion is instantaneous and should be processed separately,
4. *structure dynamics*: the interaction structure (who is talking to whom) changes over time and the agents as well as any simulated entity may be created or destroyed in the course of the simulation.

Moreover, we shall consider discrete event simulation in contrast with most existing multi-agent simulation (MAS) platform as Cormas [7], Repast [8] and many others which only consider fixed step simulations. The problem we want to tackle in this paper is to provide an implementation platform with a clear operational semantics, providing the above mentioned features, in which to map multi-agent systems, possibly combined with any other formalism.

Most, if not all, existing MAS platforms produce simulation results which do not depend only on the model but on the way the model is implemented and the scheduling ordered. This ordering is at worst arbitrary and at best randomized. We argue that it is an undesirable state of affairs and we propose to use a *DEVS*-inspired formalism to tackle this issue. *DEVS* was proposed by Zeigler [2] as a formalism to account for any kind of discrete event system. Since then a lot of *DEVS* extensions have already been proposed to handle concurrency [2] as well as structure dynamics. Among the later extensions, we can cite DS-*DEVS*[10], Dyn*DEVS*[11] and, more recently ρ-*DEVS*[3]. Some of these extensions have been proposed as tools for formalizing multi-agent systems [12,13]. Howevere, as far as we know, no systematic account still exists. In this paper, we propose to address each of the desired features by proposing either an existing or a new extension of the *DEVS* formalism, showing how it can be used to take the feature into account. One of our main contribution to *DEVS* is a clear distinction between the physical transitions simulating time-dependent physical processes and the computation of the consequences which can be both structural and informational.

Each of the following sections shall present one of the retained features, using a simple example to illustrate the use of our proposed extensions. Finally, we shall present an abstract algorithm for all the proposed extensions before concluding.

2 The Example

To illustrate our proposal, we shall take the example of the firemen fighting against fire spreading in a landscape (inspired from [14]). We have to represent the landscape made of empty spaces and forests with its dynamics of fire occurrence and spreading. We assume that the fire occurs spontaneously with a given (low) probability. The firemen are wandering around in the landscape. If a fire occurs, they are informed of the direction relative to their position and they move towards the fire. When on a burning place, they water the surface.

The model is composed of three parts:

- S a space composed of cells C with either a Von Neuman (i.e. 4 neighbors) or a Moore (i.e. 8 neighbors) topology. Each cell can be either empty, with trees, watered, burning or burned. A cell with trees has a given probability to spontaneously burn;
- T a team of firemen F positioned on the cells and whose objective is to stop the fire;
- P be the position relation linking the cells of S to the firemen of T and reciprocally.

More than one team could be defined by extending P for handling several teams. Together, P, T and P define the global structure of the system we want to stimulate.

3 Reactive-Proactive Behavior and *DEVS*

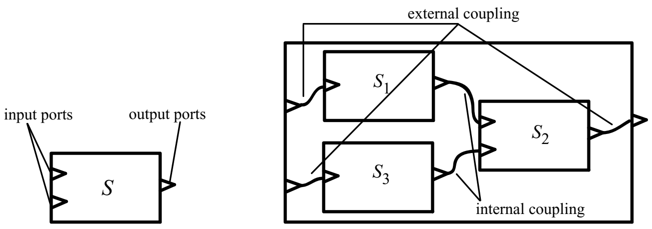
DEVS [2] is a formalism based on discrete event simulation able to model:

- *atomic entities* with a set of events incoming and outgoing through input and output ports and a set S of internal states transiting in response to incoming events (reactive behavior) or spontaneously, after having sent outgoing events (proactive behavior) (see figure 1(a)), A *DEVS entity* is a tuple $< X, Y, S, \delta_{ext}, \delta_{int}, \lambda_{ext}, \tau >$ where:
  - X is a set of input events;
  - Y is a set of output events;
  - S is a set of states;

- $\delta_{ext} : Q \times X \rightarrow S$ is the *external transition function* implementing the reactive behavior, Q is S^+ composed of $s \in S$ and the duration since the last transition;
- $\delta_{int} : S \rightarrow S$ is the *internal transition function* implementing the proactive behavior;
- $\lambda_{ext} : S \rightarrow Y$ is the *external output function* only called before an internal transition;
- $\tau : S \rightarrow \Re^+$ is the *time advance function* giving the duration until the next output and then internal transition occurrence.

Notice that the output depends on the state before the internal transition and therefore is considered as a (delayed) answer to the previous state transition, mimicking delays in physical systems.

– *composed entities* recursively composed of entities coupled through their ports and with the composed entity input and output ports (see figure 1(b)). This compositionality is defined as a *closure under coupling* property of the *DEVS* formalism.



(a) Atomic entity (b) Composed entity

Fig. 1. Atomic and composed entities

The operational semantics of an atomic entity is defined by an abstract algorithm embedded in a *simulator*. The operational semantics of a composed entity is defined by an abstract algorithm embedded in a *coordinator* which appears as a *DEVS* atomic entity for a recursively embedding composed entity. *DEVS* and any of its extensions must define the atomic entity structure, the composed entity structure and the related simulator and coordinator, obeying to the closure under coupling property. Although one can think of the simulation system as a tree of coordinators with simulators as leaves, the actual implementation often flattens the tree with a single coordinator in charge of a set of simulators. This single coordinator is then called a *DEVS-bus*.

A *DEVS*-bus computes the nearest date when an output and internal transition occurs using τ of each composing entity and then executes all the outputs, the internal transitions planned at that date and the consequent external transitions. The immediate limitation of the so-called pure *DEVS* is its arbitrary way to handle simultaneous events. When events are arriving simultaneously to the input ports of an entity, possibly simultaneous to an internal transition, the order of execution is controlled by a tie-breaking function defined in the coordinator.

3.1 Application to Multi-agent Systems

From an agent perspective, δ_{int} accounts for the proactive behavior whose occurrence is autonomously decided by the agent (τ), δ_{ext} accounts for its reactive behavior. In all cases, it is up to the agent to decide how to manage a potential conflict among the incoming events. No hypothesis is made on how these functions are computed. In particular, they could be simple condition/action rules up to sophisticated BDI reasoning, including probabilistic algorithms or not. Moreover, an agent is not necessarily a single *DEVS* entity but could be a composition of them where some are devoted to perception, others to action, etc.

4 Concurrency and *//–DEVS*

4.1 Introducing *//–DEVS*

//–DEVS [2] is an extension of *DEVS* where simultaneous events are transmitted together to the entity and the co-occurrence with an internal transition is signaled specifically. We shall briefly present *//–DEVS* before arguing its usefulness for multi-agent systems. An *//–DEVS entity* is a tuple $< X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda_{ext}, \tau >$ where:

– X is a set of input events;
– Y is a set of output events;
– S is a set of states;
– $\delta_{ext} : Q \times X^b \rightarrow S$ is the *external transition function* where X^b is the set of bags over elements in X;
– $\delta_{int} : S \rightarrow S$ is the *internal transition function*;
– $\delta_{con} : S \times X^b \rightarrow S$ is the *confluent transition function*, subject to $\delta_{con}(s, \phi) = \delta_{int}(s)$;
– $\lambda_{ext} : S \rightarrow Y^b$ is the *external output function*;
– $\tau : S \rightarrow \Re^+$ is the *time advance function.*

As in *DEVS*, τ tells the simulator and thus the coordinator the duration before an internal (proactive) transition shall occur. When the duration elapses, a bag of output events (Y^b) – throughout the paper, the upper b stands for a bag, i.e. a set of elements with duplicates – is produced by λ_{ext} – called simply λ by the *DEVS* literates – and then the internal transition occurs (δ_{int}). A bag of external events (X^b) can occur anytime before $\tau(s)$ elapsed, producing an external transition (δ_{ext}) which depends on the dynamical state in Q. Finally, if the arrival of a bag of input events occurs exactly when $\tau(s)$ elapsed, the confluent transition function(δ_{con}) is called.

The coordinator is in charge to iteratively:

1. ask to each simulator the duration τ until the next internal transition,
2. let *IMM* be the set of simulators with the same minimum duration,
3. advance the global time by that duration,
4. call λ_{ext} for each simulator in *IMM*,

5. decide for each simulator whether it should do an internal (in *IMM* with no incoming events), an external (only incoming events) or a confluent transition (both in *IMM* and with incoming events).

This algorithm guarantees that all the possible simultaneously incoming events when a simulator is in a given *state* shall be known and transmitted as a bag. However, if the duration to the next internal transition is 0 (i.e. $\tau(s)$ can be 0), the simulators can make several state transitions at the same *date*. In the next section we propose to manage when $\tau(s) = 0$ and when $\tau(s) \neq 0$ separately to distinguish between the simulation of physical transitions (which take time) and what is mere information propagation.

4.2 Application to Multi-agent Systems

In [1], we argue that action in multi-agent systems requires a special attention. The view of action as a state transition does not resist to the fact that the environment or the other agents could actually not perform the expected state transition, nor does it resist to concurrency, i.e. to the fact that the actual environment state change results from a combination of the simultaneous actions by the agents. In [1], we propose to consider agent actions as influences of which effect depends on the receiver (the environment or another agent), in the so-called influence/reaction paradigm. The mapping of this idea into $//\!-\!DEVS$ is immediate. The entities do not change directly the other entities states (would they represent an environment or another agent) but send events to them. From now on, we shall call these events, *influences* to stress this semantics. In addition, all the simultaneous influences are given at once to the receiving entity, delegating the actual state change and possible conflict resolution to the target of these influences. This property entirely complies with the influence/reaction philosophy. Moreover, no arbitrary choice is made by the coordinator, leaving to the modeler the entire responsibility of the model behavior. It contrasts with most existing MAS platforms where the choice is made arbitrarily by the scheduler, at best by randomizing the order in which the agents are run (e.g. the possibility to randomize is part of the comparison among MAS platforms in [15]).

In our example, we shall consider the case of the cellular automaton simulating fire spreading. At any state transition which are triggered at fixed time step, either by having $\forall s, \tau(s) = cst$ or by an external clock sending ticks to the cells, we chose each cell C to communicate its state to its neighbors. It means that any cell shall either perform an internal transition simultaneously to receiving up to four influences from its neighbors (δ_{con}) or perform an external transition with a tick and the same up to four influences (δ_{ext}). However if only an external transition is triggered, then no influences are output because the influences output only occur before an internal transition. Therefore only the solution where the state transition is internally triggered is possible, the influences from the firemen and the neighbors cells producing confluent transitions. According to the influence/reaction paradigm, the next state depends on all these influences at once: the advance of time for the probability of fire and the neighbors state and firemen actions for fire diffusion. Therefore, we argue that *//–DEVS* naturally solves the influence/reaction problem.

5 Instantaneity and the Logical Influences

5.1 Introducing the Logical Influences

In the previous section, we stressed that all the simultaneous incoming influences shall be provided simultaneously to the target entity for any given state, but that several state transitions can occur at the same date if $\tau(s)$ can be 0. Clearly, a physical system never does repeated instantaneous transitions, so the question rather is the meaning of these instantaneous transitions in the DEVS implementation of the physical system simulation. We propose to illustrations of this point, one in the context of formal integration and another in the context of multi-agent systems.

In the context of formal integration, one could see a composed entity as a set of coupled differential equations, on per entity. However the value of the variable of each equation should be instantaneously known by all the differential equations it is coupled to. There are several solutions to this problem: 1) the equation solving process is driven by the transmission of the variable values; 2) the transmission of the variable value is intertwined with the step-wise solving process, i.e. one step for solving and one step for value transmission; 3) the use of $\tau(s) = 0$ for value transmission and $\tau(s) \neq 0$ for step-wise equation solving.

The same is true for observation in general and perception in particular. An agent must perceive its environment. In the *DEVS* context, either the environment send influences to the agent just after each state change, or the agents send an influence for requesting information from another agent or the environment and could get an answer to his request – In a strict asynchronous communication semantics, no answer is necessarily expected, keeping agent autonomy –. In both cases, it could be made by passing time as in the solution 1 or 2 of the differential equations case, or by making $\tau(s) = 0$ for information transmission. In the first case, we consider perception as a physical process which takes time as any real physical process (a measurement device never responds instantaneously). In the second case, information diffusion, perception, or observation are considered timeless.

Consequently, we argue that $\tau(s) = 0$ is an implementation artifact to manage information diffusion, observation and diffusion and we propose to separately handle that case. Technically, we propose: 1) to forbid $\tau(s) = 0$ for the simulation of the physical transitions; 2) to add another mechanism based on *logical influences* for information diffusion. By contrast the influences producing physical state transitions shall be called the *physical influences*.

We define a $M\text{–}DEVS^{V1}$ *entity* as a tuple

$$< X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda_{ext}, \tau, \delta_{log}, \lambda_{log} >$$

with the same definitions as *//–DEVS* plus:

– $\delta_{log} : Q \rightarrow S$ the *logical transition function*;
– $\lambda_{log} : Q \rightarrow Y^b$ the *logical output function.*

and τ defined on $\Re^+ - \{0\}$ instead of $\Re^+$.

The semantics is identical to the *//−DEVS* entity semantics but for the two additional functions. λ_{log} is called after each transition (including δ_{log}) to propagate the information about the new state, while λ_{ext} is only called before an internal transition. δ_{log} is used to make computations in response to the propagated information, changing the part of S which is just a consequence of the actual physical transition and possibly producing further information diffusion. Both functions depends on Q and they do not change the duration since the last physical transition (of course). No hypothesis is made on the order in which information is propagated unlike the physical influences. However, we keep the property that all the logical influences which must arrive simultaneously when an entity is in a given state are actually transmitted simultaneously, preserving state consistency. With this new semantics, a timeless asynchronous computation process is added to the time-dependent simulation process.

When looking again at our cellular automaton example, there is clearly two different logics:

1. a physical logics based on the time steps and the influences from the firemen,
2. an informational logics of propagation of the consequences: perception of the cell state for the firemen (possibly after a request), information about the neighbor's states for the cells.

We propose to use τ, λ_{ext} and δ_{ext} for the first one, and λ_{log} and δ_{log} for the second. Although reducible to *//–DEVS*, *M–DEVS*^{V1} introduces a clearer separation of concerns. It shall further show its expressive power when combined with structure dynamics.

5.2 Application to MAS Simulations

Many MAS platforms are written in an object-oriented language like Java[8] or Smalltalk[7] and the agent and environment behaviors are very often directly written in the corresponding language as, for example, in Cormas, Repast or Mason. Therefore, the method call mechanism is automatically provided for propagating the consequences with two main limitations. The first limitation is technical: the method call is synchronous. Although the caller could ignore the result, it closes the door to parallel implementations. The second limitation is semantical: these method calls are completely free, letting the programmer change the state of any entity in the simulated system, regardless of the time coherence. In contrast, our proposal provides a cleaner semantics preserving the system coherence with respect to time management.

6 Structure Dynamics

The most important features of multi-agent systems are:

1. the openness: the agents can appear and die dynamically during the simulation;
2. the topological dynamics: the neighborhood changes over time as a consequence of the mobility of the agents in their environment (being social or physical), hence the topology of interactions.

These features make the huge difference between multi-agent systems and a set of coupled differential equations, for example, where the coupling is fixed and the equations cannot appear and disappear. We shall collectively refer to these features as structure dynamics.

For dealing with these properties, a number of extensions to *DEVS* have been proposed. Barros in [10,16] proposes to add to each coupled model, a specific atomic entity called the *executive model*, whose state defines the set of composing entities and their topology. Uhrmacher allows in [11] each entity to specify the topology of the network in a formalism called DynDEVS, a proposal we shall follow. Notice that at that stage the closure under coupling property is no longer fulfilled and therefore, the formalism is no longer a *DEVS* extension but more a *DEVS*-inspired formalism.

In the following, we shall introduce our proposal to manage the structural changes. Thereafter, we shall introduce a last, purely cosmetic, definition before providing the algorithm.

6.1 The *M–DEVS* Entity

We define a *M–DEVS entity* as a tuple

$$< X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}\delta_{log}, \lambda_{ext}, \lambda_{int}, \lambda_{log}, \lambda_{str} >$$

with the same definitions as $M\text{–}DEVS^{V1}$ plus:

– $\lambda_{int} : S \to (\Re^+ - \{0\}) \times I$ which combines τ with the specification of what to do as an *internal influence* ($\in I$);
– $\lambda_{str} : S \to Y^b$ the *structural output function* producing structural influences.

The transformation of τ into λ_{int} is only cosmetic. The rational is to have a function to express not only when to do something τ but also what to do (something which could be encoded into S). Consequently, the modified $\delta_{int} : S \times I \to S$ depends on both the state and the internal influence.

This extension is entirely reducible to the previously defined models but facilitates the expression of the algorithms as well as making the formal notations symmetrical regarding the influence kinds. In consequence:

– λ_{int} and δ_{int} are in charge of the internal transitions using internal influences and therefore of the proactive behavior;

- λ_{ext} and δ_{ext} are in charge of the external transitions using the physical influences and therefore of the reactive behavior;
- δ_{con} is managing the simultaneity of an internal and external transition;
- λ_{log} and δ_{log} are in charge of the logical transitions using the logical influences and therefore the information diffusion;
- λ_{str} is in charge of the structural changes performed externally by the coordinator itself and using the structural influences.

Once again, the operational semantics is similar to the $M{-}DEVS^{V1}$ entity but the addition of the λ_{str} function which is called after each transition in the same way as λ_{log}. In fact these two functions are closely linked to one another because the consequences of a physical transition can be both informational through the λ_{log} function and structural through the λ_{str} function. Conversely, the structural changes may require to propagate information, in particular to initialize the newly created structures.

There is no δ_{str} function because the structural influences do not change the internal state of the entities but only creates, destroys and changes the topology among the entities. Let $\Sigma = (M, Z)$ be the simulation structure where M is the set of entities and Z encodes the topology of M as a function: $< d, p_o > \rightarrow \{< r, p_i >\}$ where d, resp. r, is the sender, resp. receiver, entity, p_o, resp. p_i, is the output, resp. input, port. We define a function *change* $: X^b \times \Sigma \rightarrow \Sigma'$ which, given a bag of structural influences x^b and a simulation structure $\Sigma = (M, Z)$, computes a new simulation structure $\Sigma' = (M', Z')$. In order to be consistent with the influence/reaction paradigm, Σ' must be uniquely defined and independent of the order in which the structural influences are considered.

We distinguish four types of structural influences:

1. creation: the influence creates a new *M−DEVS* entity and attaches it to a port;
2. linkage: the influence connects the port of an entity to the entities designated by the port of another entity;
3. removal: the influence disconnects the entities linked to a given port (without destroying them);
4. deletion: the influence destroys the entity (and all the related port connections).

Whether these operations are only applicable to the issuing entity or not is open. In our current implementation, the first and last are only allowed on the issuing entity but the linkage and removal can be made anywhere as long as there is a path of successive ports (called a *port reference*) from the issuing entity.

6.2 Application to MAS Simulation

In parallel with the comparison with other MAS platforms in section 5, most existing platforms provide the primitives for creating new agents or objects as well as changing the topology. However, most of the time, these primitives are executed immediately, letting the programmer caring about the consistency with ongoing event propagation and the order in which the agents are run. In our case, all the structural changes are deferred after the physical transitions have been carried out and these changes are carefully ordered in our implementation. It guarantees that the resulting simulation structure does not depend on the order in which the modifications are issued.

The cellular automaton S as well as the team T are examples of the use of the creation structural influences. The whole structure (i.e. a grid of cells and a population of firemen interconnected together) as illustrated in figure 2 could be created "by hand" but it would be fairly cumbersome, generated automatically from a MAS specification ("compile time" generation), or generated dynamically by having S and T creating the cells and firemen when starting the simulation ("execution time" generation). In the latter case, it is enough to have λ_{str} in S and T generating one creation structural influence for each cell and each fireman, attaching each cell and fireman to a port of S and T respectively. Of course any number of teams can be put on the cellular automata by adding further Ts using the same P to manage collisions (if two firemen cannot be at the same place). Furthermore, a team could be embedded in an arbitrary number of environments with different topologies and semantics by adding further Ss and Ps.

Let "fireman" be a port of each cell for communicating with the fireman situated on the the cell and conversely let "cel" be the port for communicating with the cell the fireman is on (as illustrated in the figure 2 for the fireman F_6), any movement should change the connections of these ports to reflect the new

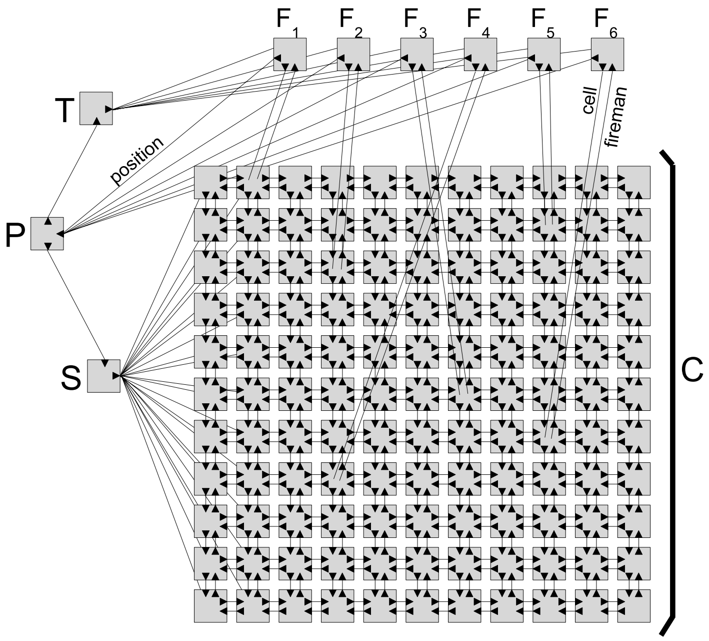


Fig. 2. The DEVS simulation structure of the example

situation. We propose to have a position P (see section 2) entity having ports to S and T respectively and performing the change using the structural influences each time a fireman moves on the grid. Accordingly, each fireman sends an influence to the position P for moving and P changes the connections among the fireman and its preceding and new cell.

Finally, when a fireman is surrounded by fire, it can also die by issuing a structural influence to delete it from the simulation. It can be done by issuing an influence to the position P to remove it from the current cell and another influence to T to remove it from the population.

Although, we provide a solution to this specific example, the mechanism is fully general and the structure could be automatically generated from higher level constructs.

7 The Abstract Algorithms

Given the various proposed extensions, it remains to formally specify the operational semantics of these constructs. As described in section 3, the operational semantics is defined by giving the abstract algorithm for the simulator (in charge of a single entity) and the coordinator (in charge of the whole simulation structure).

7.1 The *M–DEVS* Simulator

The abstract algorithm of the simulator of such a model defines the answer to five messages (instead of three for *DEVS*, see [2]). For all the messages, the variable `parent` is assumed to designate the coordinator to which the simulator is attached:

– `i-message(`t`)` to initialize the model (algorithm 1): the results are yl^b[1] and ys^b the bag of logical and structural influences to possibly build the related structure and propagate information through it. These variables of the coordinator are set as a side effect of this algorithm.

Algorithm 1. initialization of an entity

```
  $s = s_0$ {initial state}
  $t_l = t$ {initial time}
  $parent.yl^b = \lambda_{log}(s), parent.ys^b = \lambda_{str}(s)$
```

– `*-message(`t`)` to compute its output (algorithm 2): the result is the bag of the physical influences to send: ye^b;

Algorithm 2. the output of the model

```
Require: $t = t_n$
  $parent.ye^b = \lambda_{ext}(s)$
```

[1] We are using upper b as the bag notation as in the section 4.

– `x-message(`x^b`,`t`)` to compute its state transition (algorithm 3): the results are the set of logical and structural influences yl^b and ys^b as a consequence of the state change;

Algorithm 3. computation of the physical state transition

```
Require: $t_l \leq t \leq t_n$
  if $t = t_n \wedge x^b = \emptyset$ then
    $s = \delta_{int}(s)$
  else if $t = t_n \wedge x^b \neq \emptyset$ then
    $s = \delta_{con}(s, x^b)$
  else if $t < t_n \wedge x^b \neq \emptyset$ then
    $s = \delta_{ext}((s, t - t_l), x^b)$
  end if
  $parent.yl^b = \lambda_{log}(s), parent.ys^b = \lambda_{str}(s)$
```

– `l-message(`x^b`,`t`)` to compute its logical transition (algorithm 4): the results are again further logical and structural influences yl^b and ys^b.

Algorithm 4. computation of the logical state transition

```
Require: $t_l \leq t \leq t_n$
  $s = \delta_{log}((s, t - t_l), x^b)$
  $parent.yl^b = \lambda_{log}(s), parent.ys^b = \lambda_{str}(s)$
```

– finally `n-message(`t`)` to set the current date t_l, to compute the next date t_n and internal influence yi (algorithm 5).

Algorithm 5. computation of the next internal transition

```
  $d, parent.yi = \lambda_{int}(s)$
  $t_l = t, t_n = t_l + d$
```

The output influences are assumed to be of the form: $< d, p_o, y >$ where d id the entity itself, p_o is the output port and y is the influence itself. For the structural influences only $< d, y >$ is necessary.

7.2 The *M–DEVS* Coordinator

Normally in the *DEVS* philosophy, the coordinator should manage a set of coupled *M–DEVS* simulators and produce compositionally the same interface as a *DEVS* entity. In this paper, we limit ourselves to a central coordinator (a *DEVS*-bus) taking in charge the complete structure $\Sigma = \{M, Z\}$ and shall describe this algorithm in several steps:

1. given M the set of simulators in Σ, the first step consists in getting the external influences of the entities about to simultaneously perform the next internal transitions (algorithm 6). Note that ye^b is set as a side effect of calling `*-message(`t`)`.

Algorithm 6. Output before the internal transitions

```
$t = min_M(t_{n,d})$
$IMM = \{d|d \in M \wedge t_{n,d} = t\}, mail = \emptyset$
**for** $d \in IMM$ **do**
    send *-message($t$) to $M_d$
    $mail = mail + ye^b$
**end for**
```

2. the second step consists in performing the actual physical transitions, collecting the logical and structural influences (algorithm 7). IMM shall contain both the candidates for internal transition and the receivers of external influences. Note that yl^b and ys^b are set as a side effect of calling `x-message(`t`)`.

Algorithm 7. Execution of the physical transitions

```
$receivers = \{r| < d, p_o, y > \in mail \wedge < r, p_o > \in Z(d,p)\}$
$IMM = IMM + receivers, logical = \emptyset, structural = \emptyset$
**for** $r \in IMM$ **do**
    $x^b_r = \{x| < d, p_o, y > \in mail \wedge < r, p_o > \in Z(d,p)\}$
    send x-message($x^b_r$,$t$) to $r$
    $logical = logical + yl^b$
    $structural = structural + ys^b$
**end for**
```

3. the third step consists in executing the structural changes, changing both M and Z (algorithm 8 in which *change* encodes the semantics of the structural influences as described in 6.1). The newly created simulators are initialized. In 8, yl^b_r and ys^b_r designate the bag of logical influences, respectively of structural influences, computed by the corresponding simulator r by calling their `i-message` . IMM shall contain the candidates for internal transition, the receivers of external influences and the newly created entities.

Algorithm 8. Structural changes

```
$M', Z' = change(structural, M, Z)$
$structural = \emptyset$
**for** $d \in M' - M$ **do**
    send i-message($t$) to $d$
    $logical = logical + yl^b$
    $structural = structural + ys^b$
**end for**
$IMM = IMM \cup (M' - M), M = M', Z = Z'$
```

4. the logical influences are propagated (algorithm 9). yl^b and ys^b are defined as in 8 and computed when calling `l-message`. This step and the previous one are repeated iteratively until there is no structural or logical influences remaining.

Algorithm 9. Propagation of the logical influences

```
$list = logical, logical = \emptyset$
$receivers = \{r| < d, p_o, y > \in list \wedge < r, p_o, x > \in Z(d, p, y)\}$
for $r \in receivers$ do
    $x_r^b = \{x| < d, p_o, y > \in list \wedge < r, p_o, x > \in Z(d, p, y)\}$
    send l-message($x_r^b$,$t$) to $r$
    $logical = logical + yl_r^b$
    $structural = structural + ys_r^b$
end for
```

Algorithm 10. Computes next internal transitions

```
for $d \in IMM$ do
    send n-message($t$) to $d$
end for
```

5. IMM contains now all the simulators which incurred a physical transition or was newly created. These simulators are then asked to compute the date of their next internal transition for the next round (algorithm 10).

The use of a global structural *change* function hinders the possibility to distribute the model simulation. However, it is possible to overcome this limitation by composing recursively coupled $DEVS$ entities into $DEVS$ entities as illustrated in the figure 1(b). It is performed in three steps: 1) the introduction of the structure of a coupled $DEVS$ entity as a set of $DEVS$ entities and a number of input and output ports, 2) the extension of Z for coupling the input and output ports of the inner $DEVS$ entities to the input and output ports of the coupled system, 3) the restriction of the structural influences in order to only modify the structure inside of the coupled system. One obtain a variant of $DEVS$ in-between DS-DEVS[10] where a single $DEVS$ entity is devoted to structural changes and ρ-$DEVS$ [3] where all the entities can modify the structure including themselves.

8 Conclusion

This paper has presented $M{-}DEVS$ as a $DEVS$-inspired formalism for specifying the most important features of multi-agent systems, namely reactive and proactive behavior, concurrency, instantaneity and, most importantly, structure dynamics. A simple example has been used for illustrating the concepts. An operational semantics has been defined by providing an abstract algorithm to run models designed using $M{-}DEVS$. A clear distinction has been introduced between the physical transitions (using the internal and physical influences) and the computation of the consequences which can be both structural (using the structural influences) and informational (using the logical influences). These consequences are propagated until the overall structure stabilizes, before computing the date of the next physical transition.

Most, if not all, existing MAS platforms produce simulation results which do not depend only on the model but on the way the model is implemented and the scheduling ordered, this ordering being at worst arbitrary and at best randomized. We argue that it is an undesirable state of affairs and we propose a *DEVS*-inspired formalism with its algorithms which either delegates the potential ordering conflicts to the model, managing simultaneity, or orders the physical, logical and structural transitions such that the results do not depend on the issuing order, properly managing instantaneity and structure dynamics.

From the example, the resulting structure and dynamics reveal themselves as fairly complex and detailed. Actually, $M-DEVS$ should be thought as a kind of virtual machine specification for complex system simulation in which higher level specifications have to mapped. [2] has shown the possibility to map quantized dynamical systems into *DEVS*, others have mapped into *DEVS* or some of its extensions coupled differential equations and cellular automata [17]. We have illustrated the possibility to equally map multi-agent systems. In our case, an agent is mapped to a single *DEVS* entity. It could be further extended for complex agent architectures. For example, dealing with a variety of environments could require to separately specify various aspects of an agent to be coordinated, resulting in a composed architecture.

This proposal has been implemented and tested in the MIMOSA platform [18,19]. A number of applications are currently under development. From a theoretical point of view, a number of other extensions are under development like local time management, hierarchical coupling structures for holonic multi-agent systems and management of multiple points of view similar to the AGR approach [20]. The aim is to provide a formalism and its operational semantics for multi-level multi-agent systems.

I would like to thank Raphaël Duboz and my anonymous reviewers for helping me to clarify the issues, and to hopefully present it in a better way.

References

1. Ferber, J., Müller, J.-P.: Influences and reaction: a model of situated multiagent systems. In: Tokoro, M. (ed.) Proceedings of 2nd International Conference on Multi-Agent Systems, Kyoto, Japan, pp. 72–79. AAAI, Menlo Park (1996)
2. Zeigler, B.P., Kim, T.G., Praehofer, H.: Theory of Modeling and Simulation. Academic Press, London (2000)
3. Uhrmacher, A.M., Himmelspach, J., Röhl, M., Ewald, R.: Introducing variable ports and multi-couplings for cell biological modeling in devs. In: WSC 2006: Proceedings of the 37th conference on Winter simulation, Winter Simulation Conference, pp. 832–840 (2006)
4. Müller, J.-P.: Emergence of collective behaviour and problem solving. In: Omicini, A., Petta, P., Pitt, J. (eds.) ESAW 2003. LNCS, vol. 3071. Springer, Heidelberg (2004)
5. Müller, J.-P., Ratzé, C., Gillet, F., Stoffel, K.: Modeling and simulating hierarchies using an agent-based approach. In: Zerger, ndre., Argent, R.M. (eds.) MODSIM 2005 International Congress on Modelling and Simulation, Melbourne, Australia (December 2005)

6. Ratzé, C., Müller, J.-P., Gillet, F., Stoffel, K.: Simulation modelling ecological hierarchies in constructive dynamical systems. Ecological complexity 4, 13–25 (2007)
7. Le Page, C., Bousquet, F., Bakam, I., Baron, C.: Cormas: A multiagnet simulation toolkit to model natural and social dynamics at multiple scales. In: The ecoogy of scales, Wageningen, Netherlands (June 2000)
8. Tatara, E., North, M.J., Howe, T.R., Collier, N.T., Vos, J.R.: An introduction to repast modeling by using a simple predator-prey example. In: Proceedings of Agent 2006 Conference on Social Agents: Results and Prospects, Argonne, USA (2006)
9. Ferber, J.: Les systèmes multi-agents, vers une intelligence collective. InterEdition (1995)
10. Barros, F.J.: Modeling formalisms for dynamic structure systems. ACM Trans. Model. Comput. Simul. 7(4), 501–515 (1997)
11. Uhrmacher, A.M.: Dynamic structures in modeling and simulation: a reflective approach. ACM Trans. Model. Comput. Simul. 11(2), 206–232 (2001)
12. Uhrmacher, A.M.: Simulation for agent-oriented software engineering. In: Lunceford, W.H., Page, E. (eds.) First International Conference on Grand Challenges, San Diago, California (2003)
13. Duboz, R., Versmisse, D., Quesnel, G., Muzzy, A., Ramat, E.: Specification of dynamic structure discrete event multiagent systems. In: Proceedings of Agent Directed Simulation (Spring Simulation Multiconference), Hunstville, Alabama, USA (April 2006)
14. Hu, X., Muzy, A., Ntaimo, L.: A hybrid agent-cellular space modeling approach for fire spread and suppression simulation. In: WSC 2005: Proceedings of the 37th conference on Winter simulation, Winter Simulation Conference, pp. 248–255 (2005)
15. Railsback, S.F., Lytinen, S.L., Jackson, S.K.: Agent-based simulation platforms: Review and development recommendations. Simulation 82(9), 609–623 (2006)
16. Barros, F.J.: Abstract simulators for the dsde formalism. In: Medeiros, D.J., Watson, E.F., Carson, J.S., Manivanan, M.S. (eds.) WSC 1998: Proceedings of the 30th conference on Winter simulation, Los Alamitos, CA, USA, pp. 407–412. IEEE Computer Society Press, Los Alamitos (1998)
17. Wainer, G.A.: Modeling and simulation of complex systems with cell-devs. In: Proceedings of the 2004 Winter Simulation Conference, Washington DC, USA (December 2004)
18. Müller, J.P.: The mimosa generic modeling and simulation platform: the case of multi-agent systems. In: Coelho, H., Espinasse, B. (eds.) 5th Workshop on Agent-Based Simulation, Lisbon, Portugal, SCS, pp. 77–86 (May 2004)
19. `http://sourceforge.net/projects/mimosa`
20. Ferber, J., Gutknecht, O.: A meta-model for the analysis and design of organizations in multi-agent systems. In: Proceedings ICMAS 1998, Paris, France (1998)