

# Towards a logic-based, multi-agent simulation theory

Jacinto Dávila<sup>1</sup> and Kay Tucci<sup>1,2</sup>

<sup>1</sup> CESIMO (Simulation and Modelling Research Center),

<sup>2</sup> SUMA (Sistema Unificado de Microcomputación Aplicada),

Universidad de Los Andes, Mérida 5101, Venezuela

**Abstract.** This paper introduces a multi-agent simulation theory. This theory is intended to provide a formal specification to guide the development of a multi-agent simulation platform. We are extending a mature simulation language: GLIDER [DGS96] with the abstractions required to model systems where autonomous entities (agents) perceive and act upon their environments. To achieve this, we draw directly from AI mainstream research on multi-agent theories. In particular, the unified agent architecture described in [KS96] and in [Dáv97], and the model of situated multi-agent systems presented in [FM96] are employed in the extended framework. Languages of diverse nature (ranging from procedural, network-oriented to logic based programming) are, we believe, an important contribution to a multi-disciplinary approach for modelling and simulation.

**Keywords:** Multi-agent simulation, logic-based agents, influences, reactions, GLIDER.

## 1 Introduction

This paper introduces a multiagent simulation theory. This theory is intended to provide a formal specification to guide the development of a multi-agent simulation platform. We are extending a mature simulation language: GLIDER [DGS96], with the abstractions required to model systems where autonomous entities (agents) perceive and act upon their environments. In GLIDER, a system is conceived as a structured collection of objects that exchange messages. Such message exchange and processing is closely related to the scheduling and occurrence of events as in DEVS [Zei76], the conceptual framework on which GLIDER is based. Modelling a system (for simulation purposes) amounts to write a GLIDER code describing a network of nodes. Those nodes state the behaviors of the objects in the system and how, when and with which they exchange messages. GLIDER offers to the programmer a set of node types (Gate, Line, Input, Decision, Exit and Resources give its name to the language, but there are others) which the programmer instantiates to represents the objects he or she wants to simulate.

In the work presented here, we are enriching GLIDER's semantics (and syntax) to accommodate the description of agents. Agents will correspond to those entities in the modelled system that can perceive their environment, have goals and beliefs and act, according to those beliefs, to achieve those goals, presumably changing the environment in the process.

This enriching of GLIDER requires more than an additional set of language elements. We have to extend its current framework to included the behavior of the new, specialized objects: the agents. To achieve this, we draw directly from AI mainstream research on multi-agent theories. In particular, the unified agent architecture described in [KS96] and in [Dáv97], and the model of situated multi-agent systems presented in [FM96] are employed in the extended framework. In the end, we expect to have a family of languages, supported by a unique platform, to model and simulate multi-agent systems. Languages of diverse nature (ranging from procedural, network-oriented to logic based abstractions) are, we believe, an important contribution to a multi-disciplinary approach for modelling and simulation.

This paper is organized as follows: The first section offers a review of Ferber and Müller's multi-agent theory [FM96]. We recall what influences and reactions are and how they support an action model that Ferber and Müller use to found their theory. In the second section, again by referring to [FM96], we re-introduce the hierarchy of models of agents proposed by Genesereth and Nillson in

[GN88], that Ferber and Müller use to create a corresponding hierarchy of multi-agent systems. We extend those hierarchies by describing an agent that is both reactive and rational (not accounted for by Genesereth and Nilsson's work) and its corresponding multi-agent system. In the third section, we use the foundations led in the previous sections to sketch an example that illustrates an operational semantics for our family of languages: GALATEA.

## 2 A theory of influences and reactions.

In [FM96], Ferber and Müller (hereafter F-M) present a theory of multi-agent systems. They describe dynamics systems with a sort of *exhanced* state in which the universe being modelled is described via two types of "state components": *influences* and *environmental* variables. The later correspond to what is commonly known as state variables. Whereas influences are "what come from inside the agents and are attempts to modify a course of events that would have taken place otherwise" [FM96] (Pg. 73). By using influences, F-M provide for the description of concurrent events and state transitions.

Influences are a convenient way to cater for that intermediate notion of an agent "trying" to cause some change in its environment, as separated from the actual occurrence of such a change. The actual occurrence is regarded as a *reaction* of the environment to all the influences presented at a particular time. That is how they talk about influences and reactions. F-M declare that their model of action relies on three main concepts:

1. A distinction between influences and reactions, to deal with simultaneous actions.
2. A decomposition of a whole system dynamics,  $\delta$ , into two parts: the dynamics of the environment ( $\sigma$ , the *environmental state*) and the dynamics of the agents situated in this environment ( $\gamma$ , the set of all their *influences*).  $\sigma \in \Sigma$ , the set of all the possible *environmental states* and  $\gamma \in \Gamma$ , the set of all the possible sets of influences.
3. A description of the different dynamics by abstract state machines, which we use in the specification of the operational semantics of the languages in section three. Typically, an agent is characterized as tuple of attributes and a set of functions that transform that tuple. Similarly, a whole system is also characterized as a tuple (that includes its agents' tuples) and a set of transformation functions.

An important effect of F-M's approach is that, even though agents cause influences, these are not always caused by what we would call agents: Objects in the environment may become producers of influences, while having nothing else in common with agents.

The other important element in F-M's approach is the way they capture the knowledge of how, when and why the system evolves. F-M call *laws* the rules of transition towards a new environmental state, given the current environmental state and the new set of influences just generated by all the agents. Those influences have been generated by applying another type of rules: *operators*, to the previous *exhanced* state.

In the work presented here, we are taking on F-M's notions of influences and reactions and their proposal to describe dynamical systems via that *exhanced* state. However, we drop the use of *operators* and modify and extend their theory so that *laws* can be used as influence generators, among other uses. With this movement, we also establish the base for an operational semantics for our simulation language, as discussed in section three.

## 3 A hierarchy of agent architectures.

To illustrate the expressive power of *influences*, F-M adapt a classical work on agent technology to their theory. This work is Genesereth and Nilsson's hierarchy of agent's architectures [GN88]. In that

work, Generesereth and Nilsson (hereafter G-N) offered a description of a hierarchy of agent architectures ranging from a non-rational, purely reactive *TROPISTIC* agent to a rational, *DELIBERATIVE* agent, via *HYSTERECTICS* agents which are the first type of agent (in the hierarchy) with a kind of internal, “mental” state. Each type of agent is, again, modelled as a tuple which includes a number of transforming functions. The whole hierarchy from G-N, exhanced with F-M’s operators and our *REACTIVE-RATIONAL* agent is displayed in figure 1.

Agent type	Main feature
<i>REACTIVE-RATIONAL</i>	Iteratively senses, records, reasons and changes the environment
<i>DELIBERATIVE</i>	Senses, records, reasons and changes the environment
<i>HYSTERECTICS</i>	Senses, records and changes the environment
<i>TROPISTIC</i>	Senses and changes de environment
<i>OPERATORS</i>	Changes the environment

**Fig. 1.** A hierarchy of agent architectures

F-M use this hierarchy to define a corresponding hierarchy of multi-agent systems in which the agents of each type are embedded in an environment. Notice that to be faithful to their own theory F-M should have added OPERATORS as the lowest level in G-N’s hierarchy. As we explained above, OPERATORS (and in our case LAWS, as we explain below) also produce *influences* and these should be regarded as the marks of agency, because they represent what each agent tries to do.

In this paper, we will concentrate on describing just one type of agent: *REACTIVE-RATIONAL* agent. We add this one to the hierarchy with the intention of combining the best of all the other agent classes (namely, reactivity and rationality). We also hope that, by describing this agent, we will convey the fundamental notions behind this style of agent specification and multi-agent system specification.

### 3.1 Three views on perception

To explain an agent model, we need first to clarify the notions of perception and internal state proposed by G-N and the changes introduced by F-M. To model the notion of perception, G-N use a function  $Perception_a: \Sigma \rightarrow P_a$ , which maps the set of possible states of the environment ( $\Sigma$ ) to a set of percepts ( $P_a$ ) for each agent  $a$ . In G-N,  $P_a$  is a partition of  $\Sigma$ . For instance, if  $\Sigma = \{(on, hot), (on, cold), (off, hot), (off, cold)\}$ , they could have  $P_a = \{on, off\}$  meaning that agent  $a$  can only distinguish two different states of the environment through two corresponding percepts. Interestingly, F-M comment that this “realistic point of view” supposes that the agents are directly concerned with the whole state of the environment. However, one can see above that the agent has no access to certain properties of the world and, therefore, it cannot be concerned with them. Moreover, it could well be  $P_a = \{on\}$  above, in whose case the agent cannot even perceive a “partition” of  $\Sigma$ , but just a portion of it.

None the less, F-M adopt a different perception function from that of G-N,  $Perception_a: \Gamma \rightarrow P_a$ , which maps the set of all the possible *influences* to the perceptors. F-M claim that, by doing this, they preserve the separation between “influences and reactions” and produce a model which “includes automatically the locality of perception”: “Agent perceive what influences them and are not influenced by the whole state of the environment”. For this to be true, though, influences would have to annotate somehow not “the source of the influence” (the agent that causes it) but its “destination” (including the agents that are affected by that influence). F-M give no indication of the actual way of representing those influences.

Our perception function has, as F-M’s, the form  $Perception_a: \Gamma \rightarrow P_a$ . However, we accompany that function with a prescription of the representation to allow a agent to perceive what is in  $\Sigma$ .

This form is very convenient at the representational level because allows us to state that the agent perceives both the static environment (modelled by  $\Sigma$ ) and the dynamic component of the system (modelled by  $\Gamma$ ). For example, with the  $\Sigma$  above and with  $\Gamma = \{turn\_on, turn\_off\}$ , one could have  $P_a = \{obs\_on, obs\_-, off, turn\_on, turn\_off\}$ , meaning that agent  $a$  can perceive the light on, the light off **and** the actions of turning the light on and off. The *laws* (introduced below), let the modeller link  $\Sigma$  to the extended set of influences.

Another important change (w.r.t. F-M), which we expand below, it is that influences will not be linked to one point in time only. This has effects on perception, as what the agents perceive can be configured from the whole history of past influences.

Having defined the generic perceptive mechanism, we can now concentrate on the internal details of an agent.

## 4 A reactive and rational agent.

At the top of the G-N's agent hierarchy, shown above, one find deliberative, knowledge level and hysteretic agents, described (approximately)<sup>1</sup> as a 5-tuple:

$$\langle P_a, S_a, Perception_a, Memorization_a, Decision_a \rangle \quad (1)$$

where  $P_a$  and  $Perception_a$  are define as at the beginning of the previous section. The term  $S_a$  represents the set of internal states of the agent  $a$ . And the functions  $Memorization_a : P_a \otimes S_a \rightarrow S_a$  and  $Decision_a : S_a \rightarrow \Gamma$  specify: 1) how that agent  $a$  registers, in its memory, what it perceives (therefore changing its internal state) and 2) how it decides, taking into account that memory, what it will be trying to do (upon its environment) in its immediate future. It is worth noticing that the main “elements” proposed for an agent description are: The perceptual mechanism, the memorization mechanism and the reasoning engine.

F-M then take the agent so described and “places” it into the evolving dynamical system described by mean of a tuple containing: An infinite recursive function  $Evolution : \Sigma \otimes \Gamma \rightarrow \tau$  which describes how the system progresses through (infinite) time (the range of the function is a set of impossible values denoted by  $\tau$ ) and a cycle function  $Cycle : \Sigma \otimes \Gamma \rightarrow \Sigma \otimes \Gamma$  which describes how the whole system changes from one dynamical state to the next one.  $Cycle$ , in turn, is defined by means of two more functions:  $React$ , which generates the new static state (from the previous state, the previous set of influences and the laws that we explain below) and  $Exec$ , which generates a new set of influences (from the whole dynamical state and operators, which we do not need to discuss further here). Finally, agents also contribute to this new set of influences by means of a function  $Behaviour$ , specific for each agent and which characterizes its internal working and produces its influences. This collection of tuples and functions constitute a whole multi-agent system description.

We should say at this point that we preserve almost all the previous structures (only operators are deprecated). We, however, reorganized the basic structures and introduce a few new elements to helps us combine the features of *rationality*, those agent already have, with the *reactive* features we want them to have.

### 4.1 Our reactive and rational agent

We describe an agent as a 6-tuple:

$$\langle P_a, K_a, G_a, Perception_a, Update_a, Planning_a \rangle \quad (2)$$

<sup>1</sup> This is for hysteretic agent, but the points are equally illustrated.

where  $P_a$  and  $Perception_a$  are defined as at the end of section 3.1. The set  $K_a$  and  $G_a$  roughly correspond to  $S_a$  above. We want to state that a rational agent has a knowledge base,  $K_a$ , and a set of goals (or intentions),  $G_a$ , that, together, characterize its internal state.  $Update_a : \mathfrak{S} \otimes P_a \otimes K_a \rightarrow K_a$  takes the place of  $Memorization_a$  in the memorization mechanism but it now has to ensure that the addition of new information preserves the internal structure of the knowledge base (and its consistency) because  $K_a$  is a collection of logical formulae with a well-defined syntax and semantics. Similarly,  $Planning_a : \mathfrak{S} \otimes \mathfrak{R} \otimes K_a \otimes G_a \rightarrow G_a \otimes \Gamma$ , substitutes the function  $Decision_a$  and, instead of just producing influences from the internal state, the new reasoning function derives new goals and influences, taking into account the previous goals and the knowledge base. Notice that both  $Update_a$  and  $Planning_a$  introduce an argument (with domain  $\mathfrak{S}$ , the set of all the possible time points) to indicate the time at which each process (updating and planning) takes place. The introduction of time is another major change in our proposal with respect to F-M (and G-N).

With  $Planning_a$ , we want to model the process by means of which an agent derives, from a set of high level goals, a set of lower level goals, some of which are actions that can be tried for execution. This view of an agent reducing goals to subgoals has been studied in [KS96] and [Dáv97] in the context of agents in logic programming. We are using this logical model of an agent as the base of our proposal. This model also specifies a way to deal with the problem of bounded rationality, a key element of our proposal to build a reactive agent.

## 4.2 An agent with bounded rationality.

Bounded rationality refers to the fact that an agent has limited resources, typically time or memory space, to reason. The concept was used by Simon [Sim] as part of an attempt to model people as agents in an economy. He said that the perfect rational man, proposed by traditional economic models, does not represent the behaviour of real human beings because these do not reason and act that way. Human beings are influenced by a number of other variables, such a timing, and normally do not display the mathematically perfect behaviour of those models.

There have been several, recent attempts to model an agent with bounded rationality [HMP92]. Our own proposal is part of the model of an agent presented in [Dáv97]. It basically says that an agent must interleave reasoning and acting, so there must exist time (or space) bounds for the reasoning and, then, it may be that the agent acts with no-completely-refined reasons. Should it has more time to reason, the agent might take another course of actions. The key idea is that this kind of agent will be ready to (*re*)act sooner than an agent that tries to complete its reasoning process. The price to pay is that our “reactive” agent may not always take the “best” course of action.

Formally, we translate that limit into a restriction to the time the agent allocate to reasoning. Once it reaches that limit, it must switch its “locus of control” and try whatever action it had decided (if any) after that “truncated”, reasoning process. The reader may have notice that the second argument of the planning function  $Planning_a : \mathfrak{S} \otimes \mathfrak{R} \otimes K_a \otimes G_a \rightarrow G_a \otimes \Gamma$ , is a real number ( $\mathfrak{R}$ ): the time allocate to the planning (reasoning) process. In the first approximation, however, that number is an integer and it counts the number of “step of reasoning” accomplished during the last slice of time conceded to reasoning. This extension, together with the structure of the function that describe the behaviour of the agent (described immediately below) is our first proposal to model agents with bounded rationality.

## 4.3 The behaviour of an agent as a mathematical function.

Following F-M, we characterize an agent  $a$  as a mathematical function  $Behaviour_a : \mathfrak{S} \otimes \mathfrak{R} \otimes K_a \otimes G_a \otimes \Gamma \rightarrow K_a \otimes G_a \otimes \Gamma$  that maps the resource limits, the agent internal state and the set of influences to a new internal state and a set of influences produced by this agent. Unlike, F-M, of course, our agent internal state contains a knowledge base and a set of goals, as we described above.

This function  $Behaviour_a$  is defined as follows:

$$Behaviour_a(t, r_a, k, g, \gamma) = \langle k', g', \gamma' \rangle \quad (3)$$

where

$$k' = Update_a(t, Perception_a(\gamma), k) \quad (4)$$

$$\langle \gamma', g' \rangle = Planning_a(t, r_a, k', g) \quad (5)$$

The  $Update_a$  function will simply add the set of percepts observed by agent  $a$  into its knowledge base. In particular,  $obs(P, t)$  could stand for the fact that the agent observed the property  $P$  at time  $t$ .

The  $Planning_a$  function is more complicated. It specifies an inference engine which transforms goals  $g$  into goals  $g'$  and influences  $\gamma'$ , using the rules and factual information in  $k'$ , starting at time  $t$  and taking no more than  $r_a$  units of time to do it. In [Dáv97], we describe, in details, a logic program (the *demo* predicate) which could be seen as an implementation of this function. We give some further comments on the structure of that solution in the following sections.

## 5 A multi-agent rational system: a specification for a simulation language.

Up until now, we have been describing one agent. To specify the behaviour of a multi-agent system, we need to define the functions that account for the evolution of the whole system dynamics. Let us, therefore, define  $Evolution : \mathfrak{S} \otimes \Sigma \otimes \Gamma \rightarrow \tau$  and  $Cycle : \mathfrak{S} \otimes \Sigma \otimes \Gamma \rightarrow \mathfrak{S} \otimes \Sigma \otimes \Gamma$ , the same kind of functions introduced by F-M (see above), but each with a new argument to represent time:

$$Evolution(t, \sigma, \gamma) = Evolution(Cycle(t, \sigma, \gamma)) \quad (6)$$

where

$$Cycle(t, \sigma, \gamma) = \langle t', \sigma', \gamma' \rangle \quad (7)$$

where, in turn,

$$\langle \sigma', \gamma' \rangle = React(Sequence, Sequence, \mathbf{scan}, \sigma, \gamma, t) \quad (8)$$

$$Sequence = Select(Network, \xi) \quad (9)$$

$$\xi = NextEvent(\gamma) \quad (10)$$

$$t' = TimeOf(\xi) \quad (11)$$

In what follows, we explain what the  $React$ ,  $Select$  and  $NextEvent$  functions are.

### 5.1 A (new) React function (Laws as influence generators).

The key function in this description is  $React$ . In F-M,  $React$  is the overall transition function that describe how the environment changes, according to its current state, the current set of influences and certain *laws of change*. These *laws* are, precisely, a set of *domain specific rules* that the modeller introduces in a model of a certain system in order to specify the conditions and mechanism the change that system.

In our proposal,  $React$  is, again, the transition function that processes the *laws of change*. This time, however, we allow the  $React$  function to generate, not just the new static state, but a new set of influences, i.e. the whole system dynamics. What we are doing is combining the function  $React$  and  $Exec$ , from F-M's proposal, into one and dispensing with their notion of *Operators*. Our laws capture the function of both laws and operator in F-M's design, without any lost of generality and with some advantage, as we explain below. So, now laws are, as operators in F-M, *influence generators*.

Defining *Laws* as the set of all the possible laws of change, we have  $React : (Laws \cup \{; \}) \otimes (Laws \cup \{; \}) \otimes \{\mathbf{scan}, \mathbf{noscan}\} \otimes B \otimes \mathfrak{S} \otimes \Sigma \otimes \Gamma \rightarrow \Sigma \otimes \Gamma$ , defined as follows:

$$React(\varepsilon, Laws, \mathbf{noscan}, \beta, t, \sigma, \gamma) = \langle \sigma, \gamma \rangle \quad (12)$$

and

$$React(\varepsilon, Laws, \mathbf{scan}, \beta, t, \sigma, \gamma) = React(Laws, Laws, \mathbf{noscan}, \beta, t, \sigma, \gamma) \quad (13)$$

where  $\varepsilon$  is an empty sequence of laws and **scan** and **noscan** are flags' values and

$$React((\lambda; R), Laws, Flag, \beta, t, \sigma, \gamma) = \langle \sigma'', \gamma'' \rangle \quad (14)$$

where

$$\langle \sigma', \gamma' \rangle = \left\{ \left( \begin{array}{l} Reduce(name, Toreduce, \beta, t, \sigma, \gamma) \\ \text{if} \\ \lambda = \langle name, preconds, preinfluences, Toreduce \rangle \\ \text{and} \\ preconds(\sigma) \\ \text{and} \\ preinfluences(\gamma) \\ \text{or} \\ \langle \sigma, \gamma \rangle \text{ otherwise} \end{array} \right) \right\} \quad (15)$$

and

$$Flag' = \begin{cases} \mathbf{scan} & \text{if } preconds(\sigma) \text{ or } preinfluences(\gamma) \\ Flag & \text{otherwise} \end{cases} \quad (16)$$

and

$$\langle \sigma'', \gamma'' \rangle = React(R, Laws, Flag', \beta, t, refresh(\sigma, \sigma'), \gamma \cup \gamma') \quad (17)$$

*React* is the instantaneous reaction of the environment. We could add delays but, observe, that in simulation, belated effects may be introduced by posting new events in the future event list (FEL).

$\lambda$  is a law, a set of instructions, to produce a certain set of state variables and influences, given that certain preconditions for those states and influences hold (*preconds()* and (*preinfluences()*)).

Our contribution is *Toreduce*, a fragment of code that can be reduced to a set of atomic actions that transform the systems state. This reductive strategy is very similar to the reduction of logic clauses to constraints and abductive atoms as in abductive logic programming [KS96]. This reductive strategy may also allow for the specification of the operational semantics of the language used to write the instructions in *Toreduce*, as we have done in [Dáv99].

## 5.2 The whole description of MARS and its relationship to GLIDER.

On that brief description of an reactive and rational agent and the *React* function, we can build the mathematical description of a system populated by many of such agents. The only extra devices we need are the *set of all the possible mental states of all the agents (S)* and a new definition of the function  $Cycle : S \otimes \mathfrak{S} \otimes \Sigma \otimes \Gamma \rightarrow S \otimes \mathfrak{S} \otimes \Sigma \otimes \Gamma$ :

$$Cycle(\langle s_1, s_2, \dots, s_n \rangle, t, \sigma, \gamma) = \langle \langle s'_1, s'_2, \dots, s'_n \rangle, t', \sigma', \gamma' \rangle \quad (18)$$

where

$$\langle \sigma', \gamma' \rangle = React(Laws, Laws, \mathbf{scan}, BackgroundKnowledge, t, \sigma, \gamma \cup_i \gamma_i) \quad (19)$$

and

$$\langle s'_i, \gamma_i \rangle = Behaviour_a(t, r_a, k, g, \gamma) = \langle k', g', \gamma' \rangle \quad (20)$$

where, in turn,  $s'_i$  is an abbreviation of  $\langle k'_i, g'_i \rangle$ , the knowledge base and goals of agent  $i$ , and, as before,

$$Laws = Select(Network, \xi) \quad (21)$$

$$\xi = NextEvent(\gamma) \quad (22)$$

$$t' = TimeOf(\xi) \quad (23)$$

### 5.3 GALATEA's nodes and the laws: the elevator example.

Probably the best way to introduce the semantics of the new extended GLIDER language is through a simple example. What follows is the basic layout of a GALATEA simulation model. Unfortunately, there is not enough space to show it at length. It models a system with a building that has an elevator (represented by the elevator node) that carries people up and down, the doors of the elevator at each floor (e.g. DoorAtOne) and the floors themselves (e.g. FloorOne), where people stays for a while. There is also a node representing the entrance (i.e. Entrance) to the building, that "receives" people with a frequency modelled by a statistical law (exponential, with rate indicated by intArrivalTime). An finally, there is also a node representing the exit (i.e. Exit). People in the building are represented by **messages** traveling around the nodes.

As it is common in system simulation, this NETWORK describes the system and each node describes "what happens" to the entities (messages between nodes). In our context, **each node corresponds to a law** of the system.

```

NETWORK
// A building's layout

    Entrance (I) :: IT = EXPO(intArrivalTime);

    FloorOne (R) :: STAY = 10;
                   NextFloor = UNIFI(1,4);
                   if Nextfloor = 4 then SENTO(Exit);

    DoorAtOne (G) :: if WhereElevator = 1 & DoorOpen then SENTO(Elevator);

    FloorTwo (R) :: STAY = 5 ;
                   Nextfloor = UNIFI(1,3);

    DoorAtTwo (G) :: if WhereElevator = 2 & DoorOpen then SENTO(Elevator);

    Elevator (R) ::

    Exit (E) ::

AGENTS
// Each agent is specified here

    ElevatorController (AGENT) static ::
GOALS
    ( if atFloor( M ) at T and
      on( N ) at T then (
        if N = M then ( open ; turnoff ; close starting at T )
        and if N < M then ( down starting at T )
        and if M > N then ( up starting at T ) )
    and ( if true then watchfloors at any T )
    and ( if true then watchbottons at any T )

INTERFACE
// these are the instructions to explain the effects of actions on the
// environment.
```



```

INIT
// Inicialization activities.
ACT(Entrance, 0) ;

DECL
  VAR float intArrivalTime, int nextFloor, int WhereElevator;
  MESSAGES Entrance( int whereAmI, int nextFloor );
END.

```

The NETWORK section of the code above refers to the set of laws governing this system. As in GLIDER, each entrance in the network section represents a node: a subsystem of the whole system being modelled. In GALATEA, following F-M’s theory, these nodes corresponds to *laws* that state how the system changes. Thus, the behaviour of the system depends on the *scanning* of such a network for each event’s occurrence, as shown in the mathematical specification in the previous section.

The following section, AGENTS, describe the goals and knowledge base for each type of agent. The languages used for these descriptions are **logic programming languages**, which provides for greater expressiveness and are more human friendly than other computer languages. This is particularly useful when one is specifying an agent behaviour. The formal description of those languages can be found in [Dáv97].

The remaining sections (INTERFACE, DECL, INIT) provide *background knowledge* of the system. The INTERFACE is the code that describes the actual effects of each agent’s actions on the system dynamics (not shown for lack of space). The section DECL declares the global variable (VAR) and the structure of the messages (MESS), generated at I nodes, that travel around the NETWORK. And the section INIT provides for the initialization of variables and the starting event in a simulation (ACT(Node, Time)). This is the normal layout of a simulation model (in GLIDER) now enriched with a logic-based description for each agent.

#### 5.4 GALATEA at work.

Operationally, the system will interpret all the entries in a GALATEA code as a program (plan) to guide a simulation. Declaratively, one could say that the system has  $Network = [Entrance (I) :: IT := EXPO(intArrivalTime) ; FloorOne (R) :: STAY := 10; \dots]$ , as the input to the *Select* function shown above. This function simply reorders this “list” so that it starts with the node that it is being activated. At the beginning of the simulation, the first activation instruction says ACT(Entrance, 0), so that  $Network = Laws$  at that initial time, in this example.

The *React* function will then *reduce* the *Laws* to a set of instructions to change the static component of the system ( $\sigma$ ) and to schedule **future events** (these are added to  $\lambda$ ). For instance, the initial *Laws* in our example will produce something like:

$$\lambda'' = [ \text{do}(\text{simulator}, \text{set}(IT = EXPO(\text{intArrivalTime}), t_0), \\ \text{do}(\text{simulator}, \text{ACT}(\text{Entrance}, \text{now} + IT), t_0 + 1), \\ \text{do}(\text{simulator}, \text{ACT}(\text{FloorOne}, \text{now} + 10), t_0 + 2), \dots ] \quad (24)$$

This new set of influences says that: at time  $t_0$ , the simulator sets the variable IT to a value generated by the statistical function EXPO; then, at time  $t_0+1$ , the simulator schedule the activation of the node Entrance to occurs at a time equal to the current time,  $t_0+1$  (conveniently represented by the special term “now”) plus the value of the variable IT; then the simulator schedules the activation of the node FloorOne in a similar fashion, and so on.

Thus, one ends with a declarative account of the simulator activities as it simulates the evolution of a system through time. We believe that this approach may formally convey meaning to a simulation language, like GALATEA, in the same way a similar mapping from imperative to logical descriptions provides the semantical specification of a procedural language (see [Dáv99]).

Moreover, this approach supports the integration of the simulator, which controls the evolution of the environment dynamics, with the agents. We will need more space to show this, but we are aiming at something like:

$$\lambda'' = [ \text{do}(\text{simulator}, \text{set}(\text{IT} = \text{EXP0}(\text{intArrivalTime})), \text{t0}), \\ \text{do}(\text{simulator}, \text{ACT}(\text{Entrance}, \text{now} + \text{IT}), \text{t0} + 1), \\ \text{do}(\text{simulator}, \text{ACT}(\text{FloorOne}, \text{now} + 10), \text{t0} + 2), \\ \text{do}(\text{ElevatorController}, \text{up}, \text{t0} + 3), \dots ] \quad (25)$$

which accounts not just for the simulator behaviour but for the agents' **intentions** (the `ElevatorController` intends to move the elevator up a floor at time `t0+3`).

## 6 Conclusions and further work.

In this paper, we have introduced a mathematical theory that state what multi-agent systems are and how they evolve through time. It is intended to provide the formal specification to guide the implementation of a multi-agent simulation platform that we have called GALATEA. This is a multi-language platform: we use an extension to a mature simulation language (GLIDER) to describe “the world” in which the agents are embedded (the NETWORK section in the example above). But, we also use a set of logic programming languages to specify each agent goals and beliefs (the AGENTS section).

Thus, GALATEA is a multi-agent platform with a family of language to describe systems and their agents. In forthcoming work, we intend to produce both, a formal and detailed description of all those languages (syntax and semantics) and the architecture of the simulation platform.

## 7 Acknowledgment.

We are very grateful to Mayerlin Uzcátegui, Carlos Domingo and Martha Sananes for many useful discussion.

## References

- [Dáv97] Jacinto Dávila. *Agents in Logic Programming*. PhD thesis, Imperial College, London, May 1997.
- [Dáv99] Jacinto Dávila. Openlog: A logic programming language based on abduction. In *Proceedings of PPDP'99*, Paris, 1999.
- [DGS96] Carlos Domingo, Tonella Giorgio, and Martha Sananes. *GLIDER Reference Manual*. Mérida, Venezuela, 1 edition, August 1996. CESIMO IT-9608.
- [FM96] Jacques Ferber and Jean-Pierre Mller. Influences and reaction: a model of situated multiagent systems. In *ICMAS-96*, 1996.
- [GN88] Michael R. Genesereth and Nils Nilsson. *Logical foundations of Artificial Intelligence*. Morgan Kauffman Pub., California, USA, 1988.
- [HMP92] Zhisheng Huang, Michael Masuch, and L. Pólos. Alx, an action logic for agents with bounded rationality. Ccsom report 92-70, University of Amsterdam (PSCW), 1992.
- [KS96] Robert Kowalski and Fariba Sadri. Towards a unified agent architecture that combines rationality with reactivity. In Dino Pedreschi and Carlos Zaniolo, editors, *LID'96. Logic In Databases*. Informal Proceedings International Workshop on Logic in Databases, San Miniato, Italy, July 1996. (Also at <http://www-lp.doc.ic.ac.uk/UserPages/staff/rak.html>).
- [Sim] Herbert A. Simon. A behavioral model of rational choice. *Quarterly Journal of Economics*, pages 99–118.
- [Zei76] Bernard P. Zeigler. *Theory of modelling and simulation*. Wiley-Interscience, New York, 1976.