INTERNATIONAL CONFERENCE

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

PROCEEDINGS

AMSE

Association for the Advancement of Modeling and Simulation

MODELING, SIMULATION AND NEURAL NETWORKS (MSNN-2000)

MERIDA - VENEZUELA 22 – 24 OCTOBER 2000

TOWARDS A LOGIC-BASED, MULTI-AGENT SIMULATION THEORY

JACINTO DÁVILA[1]
KAY TUCCI[2]

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 , 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] CESIMO (Simulation and Modelling Research Center),
[2] CESIMO (Simulation and Modelling Research Center), SUMA (Sistema Unificado de Microcomputación Aplicada), Universidad de Los Andes, Mérida 5101, Venezuela

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 multiagent 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 Genereseth and Nillson'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, d, into two parts: the dynamics of the environment (s, the *environmental state*) and the dynamics of the agents situated in this environment (g, the set of all their *influences*). $s \hat{I} S$, the set of all the posible *environmental states* and $g \hat{I} G$, 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 thought 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

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

Figure 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 faithfull 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 $Perception_a$: $S \rightarrow P_a$, which maps the set of posible states of the environment (S) to a set of percepts (P_a) for each agent *a*. In G-N, P_a is a partition of S. For instance, if S = {(*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 S but just a portion of it.

None the less, F-M adopt a different perception function from that of G-N, $Perception_a$: $G \rightarrow P_a$, 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": "Agents 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*$_a$: $G \rightarrow P_a$. However, we accompany that function with a prescription of the representation to allow a agent to perceive what is in S. This form is very convenient at the representational level because allows us to state that the agent perceives both the static environment (modelled by S) and the dynamic component of the system (modelled by G). For example, with the S above and with G = {*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 S 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 hysteretics agents, described (approximately)[1] as a 5-tuple:

$$< P_a, S_a, Perception_a, Memorization_a, Decision_a > \quad (1)$$

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*$_a$: $S_a \rightarrow G$ 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

[1] This is for hysteretic agent, but the points are equally illustrated.

progresses through (infinite) time (the range of the function is a set o impossible values denoted by t) and a cycle function *Cycle*: $\Sigma \otimes \Gamma \rightarrow \Sigma \otimes$ which describes how the whole system changes from one dynamic state to the next one. *Cycle*, in turn, is defined by means of two mor functions: *React* , which generates the new static state (from th previous state, the previous set of influences and the laws that w explain below) and *Exec*, which generates a new set of influence (from the whole dynamical state and operators, which we do not nee to discuss further here). Finally, agents also contribute to this new se of influences by means of a function *Behaviour*, specific for eac agent and which characterizes its internal working and produces it influences. This collection of tuples and functions constitute a whol multi-agent system description.

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

4.1 Our reactive and rational agent

We describe an agent as a 6-tuple:

$$< P_a, Ka, G_a, Perception_a, Update_a, Planning_a > \quad (2)$$

where P_a and *Perception*$_a$ are defined as at the end of section 3. The set K_a and G_a roughly correspond to S_a above. We want to stat that a rational agent has a knowledge base, K_a, and a set of goals (o intentions), G_a, that, together, characterize its internal state *Update*$_a$: $\Im \otimes \Pi_a \otimes K_a \rightarrow K_a$ takes the place of *Memorization*$_a$ in th memorization mechanism but it now has to ensure that the addition o new information preserves the internal structure of the knowledge bas (and its consistency) because K_a is a collection of logical formula with a well-defined syntax and semantics. Similarly *Planning*$_a$: $\Im \otimes \Re \otimes Ka \otimes Ga \rightarrow Ga \otimes \Gamma$, substitutes the function *Decision*$_a$ an instead of just producing influences from the internal state, the ne reasoning function derives new goals and influences, taking into accou the previous goals and the knowledge base. Notice that both *Updat* and *Planning*$_a$ introduce an argument (with domain Á, the set of all t possible time points) to indicate the time at which each process (updati and planning) takes place. The introduction of time is another maj

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 o 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 [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 $Planning_a: \Im \otimes \Re \otimes Ka \otimes Ga \rightarrow Ga \otimes \Gamma$, is a real number ($\hat{A}$): 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 $Behaviour_a: \Im \otimes \Re \otimes Ka \otimes Ga \otimes \Gamma \rightarrow Ka \otimes Ga \otimes \Gamma$ 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:

$$Behaviour_a(t, r_a, k, g, g) = \langle k^t, g^t, g^t \rangle \quad (3)$$

where

$$k^t = Update_a(t, Perception_a(\gamma), k) \quad (4)$$

$$\langle \gamma^t, g^t \rangle = Planning_a(t, r_a, k^t, g) \quad (5)$$

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^t and influences g^t, using the rules and factual information in k^t, 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 $Evolution: \Im \otimes \Sigma \otimes \Gamma \rightarrow \tau$ and $Cycle: \Im \otimes \Sigma \otimes \Gamma \rightarrow \Im \otimes \Sigma \otimes \Gamma$, 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

$$Cycle(\tau,\sigma,\gamma) = < \tau',\sigma',\gamma' > \quad (7)$$

where, in turn

$$< \sigma',\gamma' > = React(Sequence,Sequence,\mathbf{scan},\sigma,\gamma,\tau) \quad (8)$$

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

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

$$\tau' = TimeOf(\xi) \quad (11)$$

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*∪{;}) Ä (*Laws*∪{;}) ⊗ {**scan**,**noscan** } ⊗ B ⊗ ℑ ⊗ Σ ⊗ Γ→ Σ ⊗ Γ, defined as follows:

$$React(\varepsilon,Laws,\mathbf{noscan},\beta,\tau,\sigma,\gamma) = < \sigma,\gamma > \quad (12)$$

and:

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

where e is an empty sequence of laws and **scan** and **noscan** are flags' values and

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

where

$$< \sigma',\gamma' > = \begin{cases} \begin{cases} Reduce(name,Toreduce,\beta,t,\sigma,\gamma) \\ \text{If} \\ \lambda = < \text{name,preconds,preinfluences,Toreduce} > \\ \text{and} \\ preconds(\sigma) \\ \text{and} \\ preinfluences(\gamma) \end{cases} \\ \text{or} \\ < \sigma,\gamma > \text{otherwise} \end{cases} \quad (15)$$

and

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

and

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

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).*

λ 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 system´s state. This reductive strategy is very similar to the reduction of logic clauses

to constraints and abductive atoms as in abductive logic programming . 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 built 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* ($\mathcal{S}$) and a new definition of the function *Cycle*: $\mathcal{S}\otimes\mathcal{T}\otimes\Sigma\otimes\Gamma\rightarrow\mathcal{S}\otimes\mathcal{T}\otimes\Sigma\otimes\Gamma$:

$$Cycle(<s_1,s_2,\ldots,s_n>,t,\sigma,\gamma)=<<s_1',s_2',\ldots,s_n'>,t',\sigma',\gamma'> \quad (18)$$

where

$$<\sigma',\gamma'>=React(Laws,Laws,\mathbf{scan},BackgroundKnowledge,t,\sigma,\gamma\cup\gamma_i) \quad (19)$$

and

$$<s_i',\gamma_i>=Behaviour_a(t,r_a,k,g,\gamma)=<k',g',\gamma'> \quad (20)$$

where, in turn, s_i^t is an abbreviation of $< k_i^t, g_i^t >$, 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** roughly **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 GALAT code as a program (plan) to guide a simulation. Declaratively, one co say that the system has *Network* = [Entrance (I) :: IT EXPO(intArrivalTime); FloorOne (R) :: STAY : = 10; ...], as the input the *Select* function shown above. This function simply reorders t "list" so that it starts with the node that it is being activated. At t beginning of the simulation, the first activation instruction s ACT(Entrance, 0), so that *Network* = *Laws* at that initial time, in t example.

The *React* function will then *reduce* the *Laws* to a set of instructi to change the static component of the system (σ) and to schedule **futu events** (these are added to I). For instance, the initial *Laws* in o example will produce something like:

$$\lambda''=[do(simulator, set(IT = EXPO(intArrivalTime), t0),$$

$$do(simulator, ACT(Entrance, now + IT), t0+1),$$

$$do(simulator, ACT(FloorOne, now + 10), t0+2), \tfrac{1}{4}] \quad (24$$

This new set of influences says that: at time t0, the simulator se the variable IT to a value generated by the statistical function EXP then, at time *t0+1*, the simulator schedule the activation of the no Entrance to occurs at a time equal to the current time, *t0+1* (convenient represented by the special term *«now»*) plus the value of the variab IT; then the simulator schedules the activation of the node *FloorO* in a similar fashion, and so on.

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

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

$$\lambda'' = [\quad do(simulator, set(IT = EXPO(intArrivalTime)), t0),$$

do(simulator, ACT(Entrance, now + IT), t0+1),
do(simulator, ACT(FloorOne, now + 10), *t0+2),*
do(ElevatorController, up, t0+3), $\frac{1}{4}$] (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 with 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. *GLID Reference Manual.* Mérida, Venezuela, 1 edition, August 1996. CESI IT-9608.

[FM96] Jacques Ferber and Jean-Pierre Müller. Influences a reaction: a model of situated multiagent systems. In *ICMAS-96*, 19

[GN88] Michael R. Genesereth and Nils Nilsson. *Logical foundati of Artificial Intelligence.* Morgan Kauffman Pub., California. USA, 19

[HMP92] Zhisheng Huang, Michael Masuch, and L. Pólos. Alx, an acti logic for agents with bounded rationality. Ccsom report 92-7 University of Amsterdam (PSCW), 1992.

[KS96] Robert Kowalski and Fariba Sadri. Towards a unified age architecture that combines rationality with reactivity. In Dino Pedres and Carlos Zaniolo, editors, *LID'96. Logic In Databases.* Inform Proceedings International Workshop on Logic in Databases, San Miniat Italy, July 1996. (Also at http://www-lp.doc.ic.ac.uk/UserPages/staf rak.html).

[Sim] Herbert A. Simon. A behavioral model of rational choic *Quarterly Journal of Economics*, pages 99-118.

[Zei76] Bernard P. Zeigler. *Theory of modelling and simulation.* Wile Interscience, New York, 1976.