A Theory for Simulations with Learning Agents

Jacinto Dávila[1], Mayerlin Uzcátegui[1,2] and Kay Tucci[1,2]
Universidad de Los Andes
Mérida, 5101. Venezuela

[1] CESIMO. ULA - jacinto@ula.ve
[2] CESIMO–SUMA. ULA - maye@ula.ve,kay@ula.ve

ABSTRACT This paper discusses a simulation theory with learning agents which is serving as a formal specification to guide the development of a multi-agent simulation platform. We have extended an existing simulation language: GLIDER, with abstractions to model systems where autonomous entities (agents) perceive and act upon their environments. We are now applying it to the study of multi-agent systems. In particular, an implementation on Biocomplexity is briefly discussed in the paper. We also show how an Inductive Logic Programming system can be used to learn rules in a representation very close to the one used to guide the simulation in the biocomplex system. This establishes the feasibility of embedding (resource-bounded) learners as agents that take part in simulating a complex system, as defined by the theory.

KEYWORDS

Multi-agent simulation

1 Introduction

This paper discusses a multi-agent simulation theory which is serving as a formal specification to guide the development of a multi-agent simulation platform [9,4].

We started by extending existing simulation language: GLIDER [1],[8], with abstractions to model systems where autonomous entities (agents) perceive and act upon their environments. Those abstractions are based on the simulation theory and provide the semantics for a new family of multi-agent, simulation languages.

For mathematicians, a theory is *"A collection of propositions to illustrate the principles of a subject"* (Oxford Dictionary). In the more accepted *simulation theory* [2], one finds a general explanation of what a system is, its components and its transitions rules, stated all as a collection of formalized, mathematical propositions. The goal of [2] and the others with that formalization (.ibid), besides supporting the explanations that are expected from a theory, was to provide the developers of *systems simulators* with a specification that says what a simulator must do and how it must behave to simulate a system.

The multi-agent, system-simulation theory discussed in this paper has exactly those purposes and is specified with similar style. This theory has served as the basic specification for the computing simulation platform GALATEA [8,9,6,3].

The papers is organized as follows: The section 2 briefly describes the basic simulation framework: GLIDER. The section 3 offers a review of Feber and Müller's multi-agent theory [12] (hereafter F&M). In section 4, we present an abstract machine as the specification of a learning agent type and briefly explain how this specification has been integrated into the multi-agent simulation theory. We also develop, in section 5, the mathematical description of a multi-agent, rational system which serves as the specification for the simulation platform. In section 6, we sketch an example from the domain of Biocomplexity in which GALATEA has been used to model a system in which human and natural dynamics are combined. The final section 7 presents a learning experiment, in which an agent learns its rules of behaviour out of a simulated history of the system (an some auxiliary concepts).

2 A briefing on GLIDER

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 [2]. Modelling a system (for simulation purposes) amounts to write a code describing a network of nodes. Those nodes state the behaviours of the objects in the system and how, when and with which they exchange messages. GLIDER offers to the modeller a set of node types (Gate, Line, Input, Decision, Exit and Resources give its name to the language, but there are others) which the modeller instantiates to represents the objects she wants to simulate.

In GALATEA, we have enriched GLIDER semantics to accommodate the description of agents. Agents 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 required more than an additional set of language elements. We had to extend its current simulation framework to include the behaviour of the new, specialized objects: the agents. We aim to have a family of languages, supported by a unique computing platform, to model and simulate multi-agent systems. Languages of diverse nature (ranging from procedural, object-oriented, network-oriented to logic-based languages[10,7]) are, we believe, an important contribution to a multi-disciplinary approach for modelling and simulation, especially when they are mapped against the same explanatory device: a common theory.

3 A theory of influences and reactions

In [12] F&M present a theory of multi-agent systems. They describe dynamics systems with a sort of *enhanced* 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 comes from inside the agents and are attempts to modify a course of events that would have taken place otherwise" [12](p73). The influence concept in the theory of F&M allows to describe the concurrence of events and the transition of states.

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, δ, into two parts: the dynamics of the environment (σ, the *environmental state*) and the dynamics of the agents situated in this environment (γ, the set of all their *influences*). Σ is the set of all the possible *environmental states* and Γ is the set of all the possible sets of influences, with $\gamma \in \Gamma$ and $\sigma \in \Sigma$.
3. A description of the different dynamics by abstract state machines, which we use in the specification of the operational semantics of the languages illustrated in section five. Typically, an agent is characterized as tuple of attributes and functions. Similarly, a whole system is also characterized as a tuple (that includes its agents' tuples) and a set of transformation functions.

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 enhanced state. However, we drop the use of *operators* and modify and extend their theory so that *laws* can be used as influence generators. With this movement, we also establish the base for an operational semantics for our simulation languages.

To illustrate the expressive power of *influences*, F&M adapt a classical work on agent technology to their theory. This work is G&N's hierarchy of agent's architectures [13]. In that work, offered a description of a hierarchy of agent architectures ranging from a non-rational, purely reactive TROPISTIC agent to a rational, DELIBERATIVE agent, via HYSTERETIC agents which are the first type of agent in the hierarchy with an 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, enhanced with F&M's operators and our REACTIVE AND RATIONAL agent is displayed in figure 1, the elements of this description are:

S_a: The set of states the agent may be in.
P_a: Partial descriptions of the environment.
$Actions$: The set of actions the agent might perform.
$Knowledge$ and $InternalS$: The set of possible internal states the agent may be in.
$Perception$: The agent sensory function.
Effector and *Decision*: These functions encode the agent's action-selection mechanism which decides the action the agent will execute.
$Memory$: This function encodes the agent assimilation mechanism by means of which it updates its knowledge base, with information from the environment (including the feedback obtained when the actions are tried).

4 Our reactive and rational agent

We now describe an agent as a 7-tuple which subsumes the one at the top of that hierarchy:

$$< P_a, K_a, G_a, Perception_a, \\ Update_a, Planning_a, Learning_a > \quad (1)$$

Agent type specification Main features
REACTIVE AND RATIONAL
$\langle S_a, P_a, Knowledge, Actions,$ $Perception, Memory, Decision\rangle$
Iteratively senses, records, reasons and acts, changing the environment
KNOWLEDGE LEVEL AND DELIBERATIVE
$\langle S_a, P_a, Knowledge, Actions,$ $Perception, Memory, Decision\rangle$
Senses, records, reasons and acts, changing the environment
HYSTERECTIC
$\langle S_a, P_a, InternalS, Actions,$ $Perception, Decision\rangle$
Senses, records and changes the environment
TROPISTIC
$\langle S_a, P_a, Actions, Perception, Effector\rangle$
Senses and changes the environment
OPERATOR OR COMPONENT
$\langle S_a, P_a, Actions, Effector\rangle$
Changes the environment

Fig. 1. The extended hierarchy of agent types

where P_a and $Perception_a$ are the percept's domain and the perception function explained below. 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, G_a, that, together, characterize its internal state. $Update_a : \Im \times P_a \times K_a \rightarrow K_a$ takes the place of $Memory_a$ 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, $Planning_a : \Im \times \Re \times K_a \times G_a \rightarrow G_a \times \Gamma$, 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 $\Im$, the set of all the possible time points) to indicate the time at which each process (updating and planning) takes place. The introduction of explicit 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 sub-goals has been studied in [14] in the context of agents in logic programming. This agent model also specifies a way to deal with the problem of bounded rationality. 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. We mark that limit with a resource (time or space to compute) counter. For this, the agent machine is regarded as resource bounded.

Finally, $Learning_a$ is a function which produces new, possibly partially refined, rules to be added to K_a. We explore possible ways to implement this function towards the end of this paper.

4.1 The behaviour of an agent as a mathematical function

Following F&M, we characterize an agent a as a mathematical function $Behaviour_a : \Im \times \Re \times K_a \times G_a \times \Gamma \rightarrow K_a \times G_a \times \Gamma$ that maps the resource limits for reasoning, 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, our agent internal state contains a knowledge base and a set of goals, as we described above.

The behaviour of that state-machine can be compactly described as follows:

$$< k'_a, g'_a, \gamma'_a > = Behaviour_a(t, r_a, k_a, g_a, \gamma) \tag{2}$$

$$k'_a = Learning_a(Update_a(t, Perception_a(\gamma), k_a), \ldots) \tag{3}$$

$$< \gamma'_a, g'_a > = Planning_a(t, r_a, k'_a, g_a) \tag{4}$$

where t is the current simulation time, r_a is a bound for the time allocated to reasoning in the agent, $k_a \in K_a$, $g_a \in G_a$, γ is the history of actions, this far, and γ_a is the set of actions this agent will try to execute.

The $Update_a$ function will simply add the set of percepts observed by agent a into its knowledge base. In particular, in $Perception_a$, $obs(P, t)$ could stand for the fact that the agent observed the property P at time t. As said above, the $Planning_a$ function specifies an inference engine which transforms goals g_a into goals g'_a and influences γ'_a, using the rules and factual information in k'_a, starting at time t and taking no more than r_a units of time to do it. The $Learning$ function can be specified by means of generalization and specialization operators applied to the knowledge base after the update to yield k'_a.

A *theory generalization operator* $\gamma_{\rho,g}$, employing the clausal generalization refinement operator ρ_g, could be defined as follows[20]:

$$\gamma_{\rho,g}(K) = \{K - \{r\} \cup \{r'\} | r' \in \rho_g(r) \ \wedge \ r \in K\} \bigcup \{K \cup \{r\} | r \in L_h\} \tag{5}$$

Similarly, a *theory specialization operator* $\gamma_{\rho,s}$, employing the clausal specialization operator ρ_s, could be defined as follows:

$$\gamma_{\rho,g}(K) = \{K - \{r\} \cup \{r'\} | r' \in \rho_g(r) \ \wedge \ r \in K\} \bigcup \{T \cup \{r\} | r \in L_h\} \tag{6}$$

where K is a knowledge base, or theory, to be revised, r and r' represent rules and L_h is the language from which those rules are written.

To implement these operators into the agent architecture, one could attempt to embed an existing ILP system, such as PROGOL, as we explain in the following sections.

5 A multi-agent rational system (MARS): 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 $Evolution : \Im \times S \times \Sigma \times \Gamma \rightarrow \tau$ and $Cycle : S \times \Im \times \Sigma \times \Gamma \rightarrow S \times \Im \times \Sigma \times \Gamma$, the same kind of functions introduced by F&M, but each one with a new argument representing time; where S represents the set of all the possible mental states of all the agents.

$$Evolution(t, < s_1, s_2, .., s_n >, \sigma, \gamma) =$$
$$Evolution(Cycle(< s_1, s_2, .., s_n >, t, \sigma, \gamma)) \quad (7)$$
$$s_a =< k_a, g_a > \quad (8)$$

Cycle, the function that steps from one global situation into the next, is defined as:

$$< t', < s'_1, s'_2, .., s'_n >, \sigma', \gamma' >=$$
$$Cycle(< s_1, s_2, .., s_n >, t, \sigma, \gamma) \quad (9)$$
$$< \sigma', \gamma' >= React(\Lambda, \beta, t, \sigma, \gamma \cup_a \gamma_a) \quad (10)$$

in which the newly introduced symbols are explained as follows:

- t: Current time.
- s_a: Agent a's internal state.
- σ: System "static" state: The environmental variables.
- γ: Set of previous influences on the environment.
- γ_a: Set of Agent a's new influences.
- Λ: The laws of the system.
- β: Background knowledge that supports the description of the system.

This description of the system must also include the equations:

$$\Lambda = Select(Network, \xi) \quad (11)$$
$$\xi = NextEvent(\gamma) \quad (12)$$
$$t' = TimeOf(\xi) \quad (13)$$
$$\beta = Interpret(InitDecl) \quad (14)$$

where,

$Select$ represents the process that extracts the laws of the system from the code provided by the modeller in the `NETWORK` section of a GALATEA model (illustrated in the last section within the example).

$NextEvent$ obtains the next event that will occur in the system from the list of influences. $TimeOf$ produces the time of that event. And $Interpret$, like $Select$, represents an interpreter that extracts background knowledge and initial settings of variables from the code that the modeller creates (also shown with the example).

5.1 The whole description of MARS

On that brief description of an reactive and rational agent and a modified $React$ function, we can build the mathematical description of a system populated by many of such agents. We only need to connect $React$ with the $Behaviour$ function for each agent, as follows:

$$< \sigma', \gamma' >= React(\Lambda, \Lambda, \mathbf{scan}, \beta, t, \sigma, \gamma \cup_a \gamma_a) \quad (15)$$

$$< s'_a, \gamma_a >= Behaviour_a(t, r_a, k_a, g_a, \gamma) \quad (16)$$

where, in turn, s'_a is an abbreviation of $< k'_a, g'_a >$, the knowledge base and goals of agent a. This links the influences from the agents' behaviour to the reaction of the environment and completes the definition of the multi-agent system.

6 An example: Agent Modelling of a Forest Reserve

What follows is the basic layout of a GALATEA simulation model of a multi-agent system coupled with a natural dynamics.

The model here described is an outcome of the project Biocomplexity: Integrating Models of Natural and Human Dynamics in Forest Landscapes Across Scales and Cultures (`http://www.geog.unt.edu/biocomplexity`). It aims to model and simulate land use and changes in vegetation cover as a consequence of human actions.

As it has been explained in [11,15], we have being devising a collection of toy models to cater for 1) the human dynamics, using the set of conceptual tools and data structures provided by GALATEA and 2) the environmental dynamics, by integrating a cellular automaton from the SpaSim [16] library into the actual simulator of a forest reserve. The data structures of GALATEA provide for the representation of the agents' goals, beliefs and observations, and, also, for a reasoning engine to deduce actions for each agent, according to its circumstances.

The simplified model considers several instances of "settler" agents and a lumber "concessionary" agent. For the sake of space, we will only consider here the behaviours of the first. For a complete account the reader is referred to [11].

The settler agent rules of behaviour can be put as shown in figure 2. The settlers are people of limited economical resources that arrive at the reserve aiming to improve their economical status and to get the property of the land that they get to occupy. Initially they dedicated themselves to subsistence agriculture: they just try to maximize the benefits from their occupation of the area, without considering soil exhaustion due to poor management practices, and without much regard for ecological damage. After five years, the land loses its fertility, and the settler must move to another available place (i.e. an area not under government supervision) or expand his farm by deforesting some adjacent land.

Figure 2 partially depicts a GALATEA model of this system[15]. This is the normal layout of a simulation model in GLIDER now enriched with a logic-based description for each agent[10,7].

Simulation results are portrayed as graphs (Figure 3) that show the percentage of total forest area by each of the policy scenarios (whose features are related to the behaviour of the Government Agent: Agroforestry, Forestry, Hands-off) and maps that

show the spatial distribution of land-use types obtained in each of the scenarios at each time step.

Figure 4 shows the final state of the Caparo Forest Reserve for each policy scenario. Our theory allows for modularity by means of a function $Behaviour_a$ for each agent but also a conceptually higher modularity by distinguishing the agents from the natural system of the forest reserve. Moreover, the theory also prescribes a role for learning in the system: to allow the agent to adapt themselves to their circunstances and change their rules of behaviour.

7 Agents that learn how to behave

The simulation theory, presented above, allows for one step further in the implementation of a platform that, not only simulates a system, but also helps to elicit knowledge from the simulations.

Let us depict a situation in which a "settler" agent is trying to learn how to be successful in a simplified version of one of the scenarios of the aforementioned bio-complex model. Figure 5 shows a fragment of the code provided to the learning system PROGOL[19] as background knowledge. Notice that it corresponds to a simplified, partial history with the actions of 10 agents. Each action is described by a predicate with the name of the action and the performing agent and the time of execution as arguments, similar to the ones in the rules in figure 2.

Working from the following set of learning examples of the concept "being successful as a settler agent A at time T", $successful(A, T)$[3]:

```
successful(ag8,5).
successful(ag8,6).
successful(ag7,5).
successful(ag7,6).
successful(ag6,5).
successful(ag6,6).

:- successful(ag0,5).
:- successful(ag9,5).
:- successful(ag1,5).
:- successful(ag2,5).
:- successful(ag3,5).
:- successful(ag4,5).
:- successful(ag5,5).
```

the Inductive logic programing engine PROGOL starts by producing the the following outputs:

[3] :- b stands for not b, to annotate the negative examples

```
[Generalising successful(ag8,5).]
[Most specific clause is]


successful(A,B) :- B=<B,
    settle(A,C),
    plant(A,D), expand(A,E),
    sale(A,E), cattle(A,B),
    cattle(A,E), cattle(A,F),
    C=<B, C=<C, C=<D, C=<E,
    C=<F, D=<B, D=<D, D=<E,
    D=<F, E=<B, E=<E, E=<F,
    F=<B, F=<F.
```

which says that *an agent A is successful at time B if she settles down, plants, expands, sales and do cattle-raising in the order indicated by the* $=<$ *conditions*. This shows how the system is able to generalize from examples to rules, that will be checked to see if they cover (positive and not negative) teaching examples. The final outcome of the learning process looks like:

```
...
[C:-9999,3,10000,0 successful(A,B)
    :- cattle(A,C),cattle(A,D).]
[C:-9999,3,10000,0 successful(A,B)
    :- cattle(A,C), cattle(A,D).]
[60 explored search nodes]
f=1,p=3,n=0,h=0
[Result of search is]

successful(A,B) :- sale(A,C), C=<B.

[3 redundant clauses retracted]
successful(A,B) :-
    sale(A,C), C=<B.
[Total number of clauses = 1]

[Time taken 0.060s]
```

which basically says that *Agent A is successful at B if she sales at C and C is before or at B*. Notice that this rule is produced after exploring many alternatives (60 nodes, or alternative rules, in this case) and choosing the one with the best evaluation score (not explained here for the lack of space, but related to the numbers accompanying the rules above).

The effect of negative examples is also meaningful, as shown by the following different output from an experiment with 9 positive examples, and the 7 negative examples above:

```
[7421 explored search nodes]
f=7,p=9,n=0,h=0
[Result of search is]

successful(A,B) :-
    sale(A,C), cattle(A,C).

[9 redundant clauses retracted]
successful(A,B) :-
    sale(A,C),
    cattle(A,C).

[Total number of clauses = 1]

[Time taken 5.100s]
```

Notice that, in all cases, the learner agent does not produce a whole plan (a complete program) but only points to crucial actions or conditions. This is due to PROGOL learning strategy of maximal compression[19] and it is certainly not a constraint if one builds in the learner within the simulation system, as we intend to do. On the contrary, it could be an useful strategy for learners that have to face an enormous store of information in the (simulated) history of the system.

8 Conclusions

In this paper, we have described a mathematical theory that state what multi-agent systems are and how they evolve through time. This theory is being used as formal specification to guide the implementation of a multi-agent simulation platform that we have called GALATEA. We have completed the development of a platform that implements the theory and we are now applying it to the study of multi-agent systems. We are exploring ways to extend the platform to allow for learning agents. We have shown how a ILP system can be used as a learning agent, in accord with the theory, to learn rules in a representation very close to the one used to simulate a multi-agent system. This establishes the feasibility of embedding (resource-bounded) learners as agents that take part in simulating a complex system.

Acknowledgements

We are very grateful to the CESIMO team and our students for many useful discussion. This work has been partially funded by CDCHT-ULA project I-886-05-02 and Fonacit project S1-2000000819. The work on Biocomplexity was also partially supported by an NSF Biocomplexity in the Environment grant CNH BCS-0216722.

References

1. GLIDER Development Group. *GLIDER Reference Manual, Versión 5.0*. Cesimo & IEAC, Universidad de Los Andes, Mérida, Venezuela, 2000. CESIMO IT-02-00. Available from: `http://afrodita.faces.ula.ve/Glider/`.

2. Bernard P. Zeigler. *Theory of modelling and simulation*. Interscience. Jhon Wiley& Sons, New York, 1976.
3. Jacinto Dávila, Erasmo Gómez, Klaudia Laffaille, Kay Tucci and Mayerlin Uzcátegui. Multi-Agent Distributed Simulations with GALATEA. *IEEE Proceedings of Distributed Simulation and Real-Time Applications.* In A. Boukerche, S. Turner, D. Roberts and G. Theodoropoulos (eds) IEEE Computer Society, pages 165–170, 2005. Aruba. `http://www.cs.unibo.it/ds-rt2005/`
4. Jacinto A. Dávila and Mayerlin Uzcátegui and Kay A. Tucci. A Multi-Agent Theory for Simulation. *The Fifth IASTED Internacional Conference on Modelling, Simulation and Optimization (MSO 2005)*, pages 285–290, 2005. Aruba. `http://www.actapress.com/Content_of_Proceeding.aspx?proceedingID=316`
5. Jacinto Dávila and Mayerlin Uzcátegui. Agents that learn to behave in Multi-Agent Simulations. *The Fifth IASTED Internacional Conference on Modelling, Simulation and Optimization (MSO 2005)*, pages 51–55, 2005. Aruba. `http://www.actapress.com/Content_of_Proceeding.aspx?proceedingID=316`
6. Jacinto Dávila and Mayerlin Uzcátegui. Gloria: An agent's executable specification. *Collegium Logicum. Kurt Gödel Society*, VIII:35–44, 2004. Vien, Austria.
7. Jacinto Dávila. Actilog: An agent activation language. Lecture Notes in Computer Science, 2562. Springer, 2003.
8. Jacinto A. Dávila and Mayerlin Uzcátegui. Galatea: A multi-agent simulation platform. *AMSE Special Issue 2000. Association for the advancement of Modelling & Simulation techniques in Enterprises*, pages 52–67, 2002. Lion, France.
9. Jacinto A. Dávila and Kay A. Tucci. Towards a logic-based, multi-agent simulation theory. *AMSE Special Issue 2000. Association for the advancement of Modelling & Simulation techniques in Enterprises*, pages 37–51, 2002. Lion, France.
10. Jacinto A. Dávila. Openlog: A logic programming language based on abduction. In *PPDP'99*, Lecture Notes in Computer Science, 1702. Springer, 1999. Available from: `http://citeseer.nj.nec.com/64163.html`.
11. M. Ablan, J. Dávila, N. Moreno, R. Quintero, and M. Uzcátegui. Agent modeling of the caparo forest reserve. In *EUROSIS 2003*, pages 367–372, Napoles, Italy, October 2003.
12. Jacques Ferber and Jean-Pierre Müller. Influences and reaction: a model of situated multiagent systems. In *ICMAS-96*, pages 72–79, 1996.
13. Michael R. Genesereth and Nils Nilsson. *Logical foundations of Artificial Intelligence*. Morgan Kauffman Pub., California. USA, 1988.
14. Robert A. Kowalski and Fariba Sadri. Towards a unified agent architecture that combine rationality with reactivity. In Dino Pedreschi and Carlos Zaniolo, editors, *LID'96 Workshop on Logic in Databases*, San Miniato, Italy, July 1996. Available from: `http://www-lp.doc.ic.ac.uk/UserPages/staff/rak.html`.
15. R. Quintero, R. Barros, J. Dávila, N. Moreno, Tonella G., and M. Ablan. A model of the biocomplexity of deforestation in tropical forest: Caparo case study. In C. Pahl, S. Schmidt, and T. Jakeman, editors, *IEMSs 2004*, Osnabrueck, Germany, June 2004. http://www.iemss.org/iemss2004/proceedings.
16. N. Moreno, M. Ablan, and G. Tonella. Spasim: A software to simulate cellular automata models. In *IEMSs 2002, First Biennial Meeting of the International Environmental Modeling and Software Society*, Lugano, Switzerland, 2002. Available from: `http://mistoy.ing.ula.ve/INVESTIGACION/PROYECTOS/SpaSim/SpaSim/`.
17. Niandry L. Moreno. Diseño e implementación de una estructura, para el soporte de simulación espacial en GLIDER. Master's thesis, Maestría en Computación, Universidad de Los Andes. Mérida. Venezuela, 2001. Tutor: Ablan, Magdiel.
18. Stephen Muggleton. Inverse entailment and progol. Technical report, The University of York, York, UK, 2002.

19. Stephen Muggleton and Jhon Firth. *Cprogol4.4: A tutorial introduction*. Department of Computer Sciences, The University of York, United Kingdom, 2002.
20. de Raedt, Luc. *Logical and Relational Learning*. From ILP to MRDM. Springer, 2006.
21. NSF. Biocomplexity: Integrating models of natural and human dynamics in forest landscapes across scales and cultures. http://www.geog.unt.edu/biocomplexity, 2002.
22. G. Tonella, M. Acevedo, M. Ablan, C. Domingo, H. Hoeger, and C. Sananes. The use of glider as a tool for the simulation of ecological systems. In M.H. Hamza, editor, *Proceedings of the IASTED International Conference*, number ISBN 0-88986-196-X, pages 463–367, Anaheim, California, October 1995. Acta Press.
23. Mayerlin Y. Uzcátegui. Diseño de la plataforma de simulación de sistemas multi-agentes galatea. Master's thesis, Maestría en Computación, Universidad de Los Andes. Mérida. Venezuela, 2002. Tutor: Dávila, Jacinto.

```
NETWORK
 LANDSCAPE (A) :: // SpaSim's invocation code
AGENTS
  Settler (AGENT) ::
  GOALS
    if supervised then go_elsewhere;
    if not(occupied_land), not(supervised),
      abandoned_land
    then settle_down_with_strategy_1;
    if not(occupied_land), not(supervised),
      land_is_forest_without_timber
    then settle_down_with_strategy_2;
    if not(occupied_land), not(supervised),
      land_is_forest_with_timber
    then settle_down_with_strategy_3;
    if land_does_not_produce,
      not(occupied_land_next)
    then expand;
  BELIEFS
    to settle_down_with_strategy_1 do move_in;
    to settle_down_with_strategy_2 do move_in,
       cut;
    to settle_down_with_strategy_3 do move_in,
       cut, sale_wood;
INTERFACE
// Code to explain the effects of the agents'
// actions on the environment.
INIT
// Initiation services.
  time_step := 10;
  ACT(LANDSCAPE, 0);
DECL
// Instructions to declare the data structures
// including those based on the SpaSim library

END.
```

Fig. 2. Partial view of the Caparo Model in GALATEA

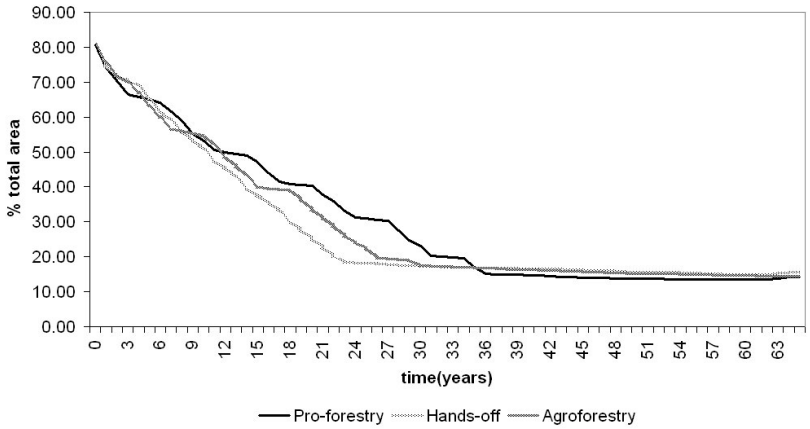

Fig. 3. Percentage of total forest area by each of the policy scenarios

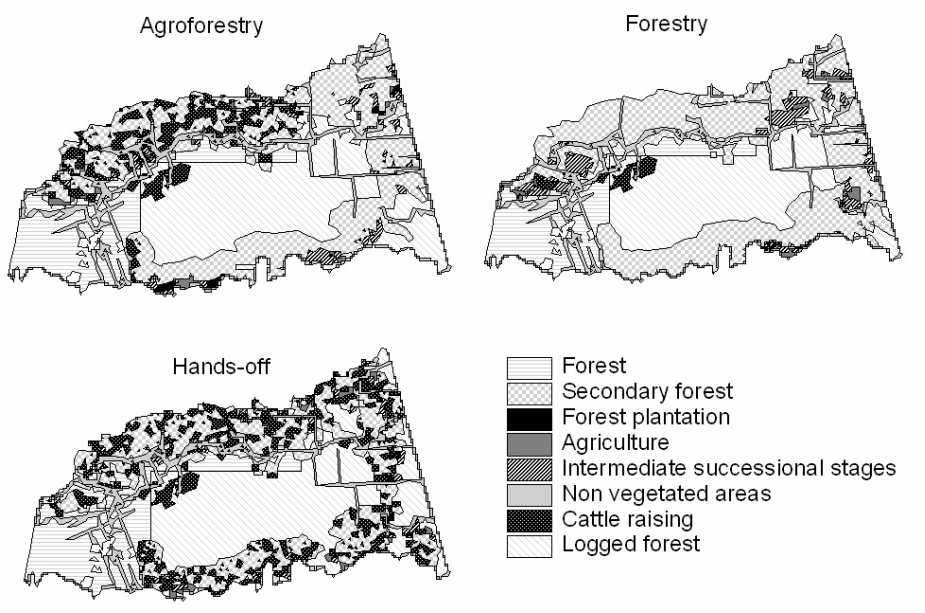

Fig. 4. Resulting maps at the end of the simulation for each one of the policy scenarios

```
% ag1
settle(ag1, 0).
plant(ag1, 0).
expand(ag1, 1).
plant(ag1, 1).
expand(ag1, 2).
plant(ag1, 2).
expand(ag1, 3).
```

```
% ag2
settle(ag2, 0).
expand(ag2, 0).
expand(ag2, 1).
plant(ag2, 2).
plant(ag2, 3).
expand(ag2, 4).
plant(ag2, 4).
plant(ag2, 5).
```

```
% ag3
settle(ag3, 0).
expand(ag3, 1).
plant(ag3, 1).
expand(ag3, 2).
plant(ag3, 2).
expand(ag3, 3).
plant(ag3, 3).
plant(ag3, 4).
plant(ag3, 5).
```

```
% ag4
settle(ag4, 1).
expand(ag4, 1).
plant(ag4, 1).
expand(ag4, 2).
plant(ag4, 2).
expand(ag4, 3).
plant(ag4, 3).
plant(ag4, 4).
plant(ag4, 5).
```

```
% ag5
settle(ag5, 0).
expand(ag5, 1).
plant(ag5, 1).
expand(ag5, 2).
plant(ag5, 2).
expand(ag5, 3).
cattle(ag5, 3).
cattle(ag5, 4).
cattle(ag5, 5).
```

```
% ag6
settle(ag6, 0).
plant(ag6, 1).
expand(ag6, 3).
sale(ag6, 3).
cattle(ag6, 3).
cattle(ag6, 4).
cattle(ag6, 5).
```

```
% ag7
settle(ag7, 0).
plant(ag7, 1).
expand(ag7, 3).
sale(ag7, 3).
cattle(ag7, 3).
cattle(ag7, 4).
cattle(ag7, 5).
```

```
% ag8
settle(ag8, 0).
plant(ag8, 1).
expand(ag8, 3).
sale(ag8, 3).
cattle(ag8, 3).
cattle(ag8, 4).
cattle(ag8, 5).
```

```
% ag9
settle(ag9, 0).
plant(ag9, 1).
expand(ag9, 3).
buy(ag9, 3).
cattle(ag9, 3).
cattle(ag9, 4).
cattle(ag9, 5).
```

```
% ag0
settle(ag0, 0).
plant(ag0, 1).
expand(ag0, 3).
cattle(ag0, 3).
cattle(ag0, 4).
cattle(ag0, 5).
sale(ag0, 6).
```

Fig. 5. A simplified history in a MAS: Agents acting in a Forest Reserve. Which one is more successful?