ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **11**, Number 3, 2008, 267–280

OPERAS for Social Insects: Formal Modelling and Prototype Simulation

I. STAMATOPOULOU[1], I. SAKELLARIOU[2],
P. KEFALAS[3], G. ELEFTHERAKIS[3]

[1]South East European Research Centre, Thessaloniki, Greece
E-mail: istamatopoulou@seerc.org

[2]Department of Applied Informatics, University of Macedonia
Thessaloniki, Greece
E-mail: iliass@uom.gr

[3]Department of Computer Science, CITY College, Thessaloniki, Greece
E-mail: {kefalas, eleftherakis}@city.academic.gr

Abstract. Social insect colonies present an interesting problem for formal modelling due to characteristics such as self-organisation and dynamic structure. In this paper, we present a formal model of a colony of Pharaoh ants using $OPERAS_{XP}$, which combines two different formal methods, communicating X-machines and population P systems as well as a framework that leads to a rapid simulation prototype of such multi-agent systems, based on their formal models.

1. Introduction

Study of social insects colonies, such as ants and bees, reveal the need for computational models which are able to handle the highly dynamic structure of any biological or nature-inspired artificial system that exhibits emergent behaviour. These systems can be directly mapped to multi-agent systems (MAS) by considering each entity as an agent, with its own behavioural rules, beliefs, goals, decision making mechanisms and means of communication with the other entities and with the environment. The overall behaviour of the system is merely the result of the agents' individual actions, the interactions among them and between them and the environment.

A key aspect that has to be dealt with at the modelling level is the dynamic nature of such MAS and how their structure is constantly reconfigured. By *structure* we

imply (i) the number of the agents, and (ii) either their physical placement in space or, more generally, the structure that is dictated by the communication channels among them. Most modelling methodologies assume a fixed, static structure that is not realistic since in a dynamic MAS, communication between two agents may need to be established or ceased at any point and also new agents may appear in the system while existing ones may be removed. Related to the above, an additional issue that the dynamic nature of these systems raises has to do with distinguishing between the modelling of the individual agents (behaviour) and the rules that govern the structural adaptation of the collective MAS, the latter referring to the part of the agent responsible for non-behavioural issues. A modelling method that allows such a distinction, would greatly assist the modeller by breaking down the work into two independent activities.

Part of the bio-informatics technology aims at developing in-silico models and simulations that will complement in-vitro and in-vivo biological experiments. On the other hand, knowledge gained from observation in these experiments could be used to develop artificial systems in which simple components with simple interactions will achieve a complex overall behaviour.

This paper presents how such a multi-agent system can be formally modelled using $OPERAS_{XP}$ and how this model may be used towards the simulation of the system in NetLogo. In the next section the in-nest behaviour of the Pharaoh ants to be modelled is briefly described. Section 3 briefly presents the $OPERAS$ framework and how it can be instantiated using communicating X-machines and population P systems, leading to $OPERAS_{XP}$. Section 4 presents how the Pharaoh ants may be modelled using the proposed method and in Section 5 we discuss the steps leading from our formal model toward a rapid simulation in the NetLogo platform. Finally, Section 6 concludes this paper.

2. Pharaoh Ants

Monomorium pharaonis, also called the Pharaoh ants, is a species of small ants that originated from North Africa. Due to their small size and their rapid reproduction cycle they are typically studied in-vitro. A typical colony of such ants comprises 100 to 5000 ants, containing a queen, a number of workers, pupae and some brood, however a number of about 100 to 200 ants is adequate to study their behaviour.

The ants spend much of their time doing nothing, thus staying inactive. An ant may become active if it becomes hungry or if it is being recruited to forage by another ant. Such interactions between ants happen within the nest and it is their behaviour inside there we attempt to model.

The assumptions that are made in this study are the following: (i) the colony is situated in a rectangular environment and only consists of workers; (ii) the ants are either inactive or hungry, moving around in search for food; (iii) when two ants meet they might share food, if one is hungry and the other has food supplies; and (iv) the ants go outside to forage when they are hungry, no food source is identified inside the nest and a pheromone trail leading to an exit of the nest is discovered, (v) the laying

of pheromone is a behaviour of the foraging ants that return back to the nest carrying food and is therefore not explicitly modelled.

Though this is a fairly simple case study, it is very realistic and of interest for various reasons. It shows a combination of independent behaviours of the ants as well as synchronised behaviour, when two ants come across to exchange food. It also has an important degree of repetitiveness using the same type of ant in a number of instances but also with slight variations between them (through the food distribution across the ant colony, their different positions in the environment and the differences in the individuals' hunger thresholds etc.). Finally, it exhibits ant colony self-organisation aspects.

From a modelling perspective, there are several interesting properties of the system that are challenging. Firstly, the individual behaviour of the ants must be modelled also enabling them to perceive their environment. The number of ants is not static, since new ants may enter the nest and some leave or die of hunger, but neither is their communication network, as pairs of ants communicate under particular conditions. Overall, the structural configuration of the colony is highly dynamic and constantly changes over time.

3. The OPERAS Framework

According to the *OPERAS* framework [15], when modelling a MAS, one should specify a number of agents, their types and abilities, the environment in which they operate, the stimuli that can be perceived by the agents, the agents' communication network and the rules that govern the structural mutation of the overall system. From this perspective, a *Multi-Agent System* model in its general form can be defined by the tuple (O, P, E, R, A, S) containing:

- The set of reconfiguration rules, O, that define how the system structure adapts by applying appropriate reconfiguration operators. The rules in O are of the form *condition* $\Rightarrow$ *action* where *condition* refers to the computational state of agents and *action* involves the application of one or more operators that create/remove a communication channel between agents or introduce/remove an agent into/from the system.
- The set of the model's percepts, P, is defined as the distributed union of the sets of percepts of all participating agents.
- The environment's model / initial configuration, E.
- The relation, R, describing the initial structure of the communication network. It is defined as $R : A \times A$ with $(A_i, A_j) \in R$, $A_i, A_j \in A$, meaning that agent A_i may communicate with agent A_j, and it changes during the computation so as to depict the communication network at any one point.
- The set of participating agents, $A = \{A_1, \ldots A_n\}$ where A_i is a particular agent defined in terms of its individual behaviour and its local mechanism for structure mutation.

- The set of type definitions of agents that may be present in the system, $S = \{(Behaviour_t, StructuralMutator_t) \mid t$ being an agent type identifier$\}$, where $Behaviour_t$ is the part of the agent that deals with its individual behaviour and $StructuralMutator_t$ is the local mechanism for structure reconfiguration; each participating agent A_i of type t is a particular instance of a type of agent: $A_i = (Beh_t, StrMut_t)_i$.

The general underlying idea is that an agent's formal model consists of two parts, its behaviour and its structural mutator. The behaviour of an agent can be modelled by a formal method with its computation being driven by percepts from the environment. The structural mutator can be modelled by a set of reconfiguration rules which given the computation states of agents can change the structure of the system. The MAS structure is determined through the relation that defines the communication between the agents. The set of participating agents are instances of agent types that may participate in the system. This deals with the fact that an agent may be present at one instance of the system but disappear at another or that a new agent comes into play during the evolution of the MAS. This assumes that all agent types that may participate in the system should be known in advance.

OPERAS$_{XP}$

There are several options for instantiating $OPERAS$ into a concrete modelling method, considering that for the modelling of each type of agent S_k, different methods may be used to specify its behavioural part and its structure mutation mechanism. We have long experimented with two formal methods, which are communicating X-machines (CXMs) and population P systems (PPSs) with active cells.

X-machines, a state-based formal method introduced by Eilenberg [3], are considered suitable for the formal specification of a system's components. More particularly, stream X-machines were found to be well-suited for the modelling of reactive systems. Since then, valuable findings using the X-machines as a formal notation for specification, communication, verification and testing purposes have been reported [8, 4, 6]. An X-machine (XM) model consists of a number of states and also has a memory, which accommodates mathematically defined data structures. The transitions between states are labelled by functions. In addition to having stand-alone X-machine models, communication is feasible by redirecting the output of one machine's function to become input to a function of another machine. The system structure of *communicating X-machines* is defined as the graph whose nodes are the components and edges are the communication channels among them [1].

On the other hand, a *population P system* [2] is a collection of different types of cells evolving according to specific rules and capable of exchanging biological/chemical substances with their neighbouring cells. The rules specifying the behaviour of the individual cells in a PPS are more commonly of the simple form of rewrite rules which are typically not sufficient for describing the behaviour of the respective agent a cell may represent. However, PPS provide a straightforward way for dealing with the change of a system's structure through division rules, for creating new cells, cell death

rules, for removing cells, and bond-making rules, for reconfiguring the communication links among cells.

Ad hoc integration of these two methods [14, 9, 13] gave us some preliminary results which led us to the current combined approach we take for *OPERAS*. It is interesting to notice that none of the two formal methods by itself could successfully (or at least intuitively) model a MAS [9, 14], however the *OPERAS* framework gives the opportunity to combine them as best suited for either of the two modelling tasks. In the following, we present an instance of *OPERAS*, named $OPERAS_{XP}$, that uses CXMs and PPSs. Previous work on instantiating the *OPERAS* framework included another version of this combined approach ($OPERAS_{XC}$ [15]) as well as a version that uses only PPSs [16].

For the following, we consider that the computation state of a CXM describing the behaviour of an agent is a 3-tuple $Q \times M \times \Phi$ representing (i) the state q_i the CXM is in, (ii) its current memory m_i, and (iii) the last function φ_i that has been applied (or ε, standing for 'empty', in the initial configuration). Though it may not be customary to include the last applied function as part of the computation state, our modelling experience has shown that knowing *how* an XM has come to a particular state can at times be valuable information. An abstract example of an $OPERAS_{XP}$ model consisting of two agents is depicted in Fig. 1.

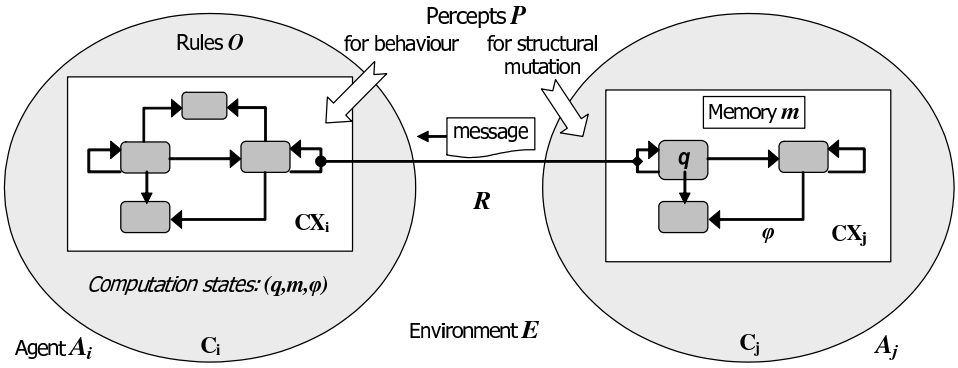

Fig. 1. An abstract example of an $OPERAS_{XP}$ consisting of two agents.

A MAS in $OPERAS_{XP}$ is defined as the tuple (O, P, E, R, A, S) where:

- the rules in O, responsible for structure reconfiguration, will be PPS cell division, cell death and bond-making rules, whose applicability will depend on the computation state (q, m, φ) of a CXM, in order to add/remove agents/communication links. These rules are of the form:
  - Cell division: $((q, m, \varphi) \rightarrow (q, m, \varphi)\ (q', m', \varepsilon))_t$ meaning that when an agent (CXM) of type t is in the computation state (q, m, φ), a new agent may be introduced in the system initially in the computation state (q', m', ε) (since for a new agent no function has been applied).

  - Cell death: $((q, m, \varphi) \rightarrow \dagger)_t$ meaning that when an agent (CXM) of type t is in a particular computation state (q, m, φ), the agent is removed from the system.
  - Bond-making: $(t_1, (q_1, m_1, \varphi_1) ; (q_2, m_2, \varphi_2), t_2)$ meaning that when two agents of type t_1 and t_2 are in the computation states (q_1, m_1, φ_1) and (q_2, m_2, φ_2), respectively, a communication link is created between them.

- $P = P_B \cup P_{SM}$ is the set of percepts of all participating agents, where $P_B = \Sigma_1 \cup \ldots \cup \Sigma_k$ is the set of inputs perceived by the XM model of the behaviour (subscript B) and $P_{SM} = (Q_1 \times M_1 \times \Phi_1) \cup \ldots \cup (Q_k \times M_k \times \Phi_k)$ is the set of objects (alphabet) in the PPS cells that deal with structural mutation (subscript SM), $k = |S|$ being the number of types of agents;

- $E = \{(q, m, \varepsilon)_i \mid 1 \leq i \leq n,\ q \in Q_i,\ m \in M_i\}$ holding information about the initial computation states of all the participating agents;

- $R : CXM \times CXM$ (CXM: the set of CXMs that model agent behaviour), depicting the initial communication network between the ants. R is updated throughout the computation by the bond-making rules;

- $A = \{A_1, \ldots, A_n\}$ where $A_i = (CXM_t, C_t)_i$ is a particular agent of type t defined in terms of its individual behaviour (CXM_t) and its local structural mutator cell for controlling reconfiguration (C_t). The structural mutator cell is of the form $C_t = (w_i, o_t)$ where w_i is the multi-set of objects it contains and $o_t \subset O$ is the set of rules that correspond to the particular type of agent, t;

- $S = \{(XT_t, C_t) \mid t$ being an agent type identifier$\}$, where XT_t is an XM type, with no initial state and memory.

In this model, each structural mutator cell implicitly knows the computation state (q, m, φ) of the underlying XM that models behaviour. Environmental input is directed straight to the agent's behavioural part. In each computation cycle an input triggers a function of the behaviour CXM and the updated information about the agent's current computation state is updated in the structural mutator cell. A copy of the object is placed in the environment for other agents in the local environment to have access to it. Objects from the environment representing the computation states of neighbouring agents are imported and finally, all the reconfiguration rules in O of the type of the particular cell are being checked and if necessary applied. Since the model follows the computation rules of a CXM system (triggered by the behaviour component's input, asynchronously for the different participating agents), computation of the behaviour-driven version of $OPERAS_{XP}$ is asynchronous.

4. $OPERAS_{XP}$ Model of the Pharaoh Ants

For the problem at hand, one initially has to create the CXM model of an ant, starting by identifying the number of states. There are five states which the ant can

be in: (a) *Inactive*, a non-hungry ant holding food, (b) *Hungry*, an ant holding a food quantity that is below its hunger threshold, (c) *Giving*, a non-hungry ant that perceives a hungry one and shares its food, (d) *Taking*, a hungry ant that perceives an inactive ant and receives food from it, and (e) *Dead*, for an ant whose food quantity has dropped to zero. Formally we write:

$$Q = \{Inactive, Hungry, Giving, Taking, Dead\}$$

The memory of the ant holds (a) its *current position*, (b) the *amount of food*, it carries, (c) a number denoting the *food quantity threshold*, below which the ant becomes hungry, (d) the *food decay rate*, a number denoting the quantity of food that is consumed by the ant in each time unit, and (e) the *food portion*, i.e. the food amount to be given by an ant that is carrying food to another which is hungry. We choose to represent all of them as natural numbers.

The ant is modelled so that it accepts a tuple $(pos, stimuli)$ as input to its functions. The first element of the input represents the coordinates in which stimuli is perceived whereas the second element is the description of the stimuli. The latter can be *pheromone* or *space*, describing an empty position with or without pheromone, a hungry, *ha*, or a non-hungry, *nha*, ant, or, finally, a number greater than zero representing the food quantity that is received by a non-hungry ant that shares its food. Formally:

$$\Sigma = (\mathbb{N} \times \mathbb{N}) \times (\{ha, nha, pheromone, space\} \cup \mathbb{N})$$

The transitions between states of the XM, as indicated by the state transition diagram in Fig. 2, are the functions:

$$\begin{aligned} \Phi \quad = \quad & \{search,\ followTrail,\ becomeHungry,\ ignoreHungryAnt,\\ & meetInactiveAnt,\ meetHungryAnt,\ die,\ doNothing,\ giveFood,\\ & noFoodToGive,\ takeEnoughFood,\ takeNotEnoughFood\}. \end{aligned}$$

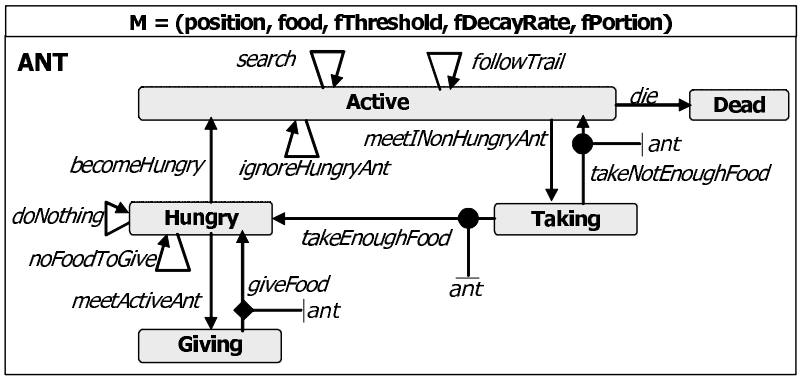

Fig. 2. The state transition diagram of the ant.

Functions are triggered by an input and the contents of the memory, and they produce an output while updating the memory. For example, when in the *Inactive* state an ant may: (a) perceive a hungry ant and if the food quantity the former is carrying

is enough, this will bring it to the *Giving* state (function *meetHungryAnt*) whereas if it does not have enough food to give, it will ignore the hungry ant and remain in the *Inactive* state (function *noFoodToGive*); (b) *becomeHungry*, if the amount of food it carries drops below its hunger threshold, or (c) *doNothing*. The only thing that an ant in the *Giving* state can do is to actually give the food to the hungry ant that has been just perceived (function *giveFood*). Note, finally, that the action of eating food is not explicitly modelled; rather, an ant consumes an amount of food equal to the decay rate at every computation step (every time a function is applied).

Indicatively, the *becomeHungry* function is in formal notation as follows:

$$becomeHungry((pos, in), (myPos, food, fThreshold, fDecay, fPortion)) = \\ ((got\_hungry), (myPos, food - fDecay, fThreshold, fDecay, fPortion)) \\ \text{if } food - fDecay \leq fThreshold$$

CXMs can communicate by exchanging messages. This is achieved by directing the output of one CXM function to become input to another CXM's function. In relation to our model, two ants are in communication while sharing food. The inactive ant's function *giveFood* sends as output (⧫ symbol in Fig. 2) the food amount it is willing to share to be received as input (• symbol in Fig. 2) by the *takeEnoughFood* or *takeNotEnoughFood* functions of the hungry ant.

The definition of the ant XM corresponds to the behavioural part of the agent whereas the structural mutator part will be a PPS cell that holds and applies rules to objects of the form (q, m, φ) (XM computation state). The remaining elements of the $OPERAS_{XP}$ definition are as follows:

The set O contains the rules that capture the ways in which the structure of the colony mutates through the appearance and removal of agents or communication links. Two kinds of PPS rules (cell division and bond-making) are required for the particular scenario (in the following, the '_' represents an element value that is unimportant for the rule).

When two ants, one of them hungry and the other one inactive, perceive each other, the functions *meetInactiveAnt* and *meetHungryAnt* are applied, bringing them to the *Taking* and *Giving* states, respectively. At this point a communication link must be created and this is accomplished by the following bond-making rule:

$$(ant,\ (Giving, (pos_1, \_, \_, \_, \_), meetHungryAnt)\ ; \\ (Taking, (pos_2, \_, \_, \_, \_), meetInactiveAnt),\ ant) \\ \text{if } neighbours(pos_1, pos_2)$$

where *ant* denotes the type of the two agents.

The following cell death rule removes an ant from the system, when its food quantity (2nd memory element) drops to zero:

$$((\_, (\_, food, \_, \_, \_), \_) \rightarrow \dagger)_{ant}, \text{ if } food \leq 0$$

The set of percepts P, since we are only dealing with one type of agent (ant), is equal to the input set Σ of the XM.

The environment E needs to hold objects of the form (q, m, φ) (or (q, m, ε) in the initial configuration) that need to be communicated among neighbouring cells through appropriate PPS communication rules, so that each ant can "know" the computation

state of its neighbour(s) (as for example required by the bond-making rule presented above).

The relation R, finally, holds pairs of ant instances that are in communication at any one point, and is being accordingly updated by the bond-making rule at the end of every computation cycle.

5. From $OPERAS_{XP}$ to NetLogo Simulation

Apart from verifying the proposed model using formal methods, the presence of a large number of XMs interacting based on spatial information dictated the need for a simulation application that would animate the execution of the proposed model. Such an animator would allow us to visualize the system's operation as well as collect statistical data concerning the colony's behaviour in the nest. Implementing such an animator requires a tool that supports simulation of a large number of autonomous entities, the Pharaoh ants in our case, as well as sufficient facilities for building a graphical representation of the environment and collecting experimental data. One of the best representatives of such tools is NetLogo.

NetLogo [17] is a modelling environment targeted for simulation of multi-agent systems that consist of a great number of agents. NetLogo offers a simple functional language, in which behaviours of agents can be encoded, and a programming environment that allows the easy creation of a GUI for a simulation supporting a great number of parameters. The environment is an excellent tool for rapid prototyping and initial testing of multi-agent systems as well as an excellent animation tool for the modelled system.

There are two types of agents in NetLogo: static agents that are called *patches* and are components of a grid on which agents that are able to move, called *turtles*, "live" and interact. The former allow the modelling of environmental properties in a simulation, such as the existence of pheromone in a specific nest position. The latter can be used to model fully capable agents such as, in our case, the ants. The programming language allows the specification of the behaviour of each patch and turtle, and of the control of the execution. Moreover each turtle can have its own set of variables, i.e. can carry its own state. The latter greatly facilitated the implementation of the nest's simulation, since each ant (turtle) in the simulation maintained its own XM memory structure and information about its state. Monitoring and execution of the agents is controlled by an entity called *observer* that "asks" each agent to perform a specific computational task.

As stated, the motivation behind building the NetLogo simulation was to test the proposed $OPERAS_{XP}$ model for the Pharaoh ants with the additional aim of performing a number of experiments to investigate the colony's behaviour in a nest that contains a large number of ants. One choice would be to build a completely ad hoc model of the ants, based on a subjective interpretation of the functions and transitions as they are described in the XM model. However, since XM models can be defined using the X-machine Description Language (XMDL), a standard notation that allows the representation of all the elements of the construct tuple of an XM [7], we have chosen to implement an XM meta-interpreter that will animate the specified

XM model from a NetLogo representation of the XMDL specification. So that the above is better illustrated, consider the following XM function (defined in XMDL):

```
#fun become_hungry ((?p, ?in), (?pos, ?f, ?ft, ?fdr, ?fp)) =
  if ?nf =< ?ft then ((got_hungry), (?pos, ?nf, ?ft, ?fdr, ?fp))
  where ?nf <- ?f - ?fdr.
```

In our simulation the above is translated into the following NetLogo function:

```
to-report become_hungry [px? py? in?]
  let nf? f? - fdr?
  ifelse nf? <= ft?
    [report(list true "got_hungry" (list xcor ycor nf? ft? fdr? fp?))]
    [report(list false)]
end
```

The functions are executed by the meta-interpreter which continuously executes the following loop:

```
1. Execute Applicable Dynamic Reconfiguration Rules
2. Ask each ant to:
   2.a Determine its input based on status of its neighbouring positions.
   2.b Determine the applicable function based on input and
       state of the ant.
   2.c Apply changes to its memory and state specified by the
       selected function of step 2.b.
```

Implementation of the structure reconfiguration rules is rather straightforward in NetLogo, given that the observer "controls" the execution of agents and has access to all their internal variables, i.e. a complete view of the state of computation, which is necessary in order to decide which of the rules are triggered.

For example, when an ant is in the state "dead", it must be removed from the system, according to the corresponding reconfiguration rule. This is implemented in the following line of code:

```
ask ants with [state = "dead"] [die]
```

In the code above, `die` is a NetLogo keyword that removes a turtle from the simulation environment. Bond-making rules are slightly more complex. A list of all possible `communication-partners` (global variable) is created by the `select-hungry-ant` procedure executed by each inactive ant, and among these some are selected for applying the bonding (thus representing the communication relation R). A code fragment is shown below (the full code further contains a number of checks that were necessary in such a dynamic model).

```
ask ants with [state = "inactive"] [select-hungry-ant]
foreach communication-partners [
  let ha first ?
  let nha item 1 ?
```

```
if state-of ha = "taking" and state-of nha = "giving"
  [ask ha [set pref-input? (list xcor-of nha ycor-of nha mfp?)]
   ask nha [set pref-input? (list xcor-of ha ycor-of ha mfp?)]]
if state-of ha = "taking" and state-of nha != "giving"
  [ask ha [set pref-input? (list xcor-of nha ycor-of nha 0)] ] ]
```

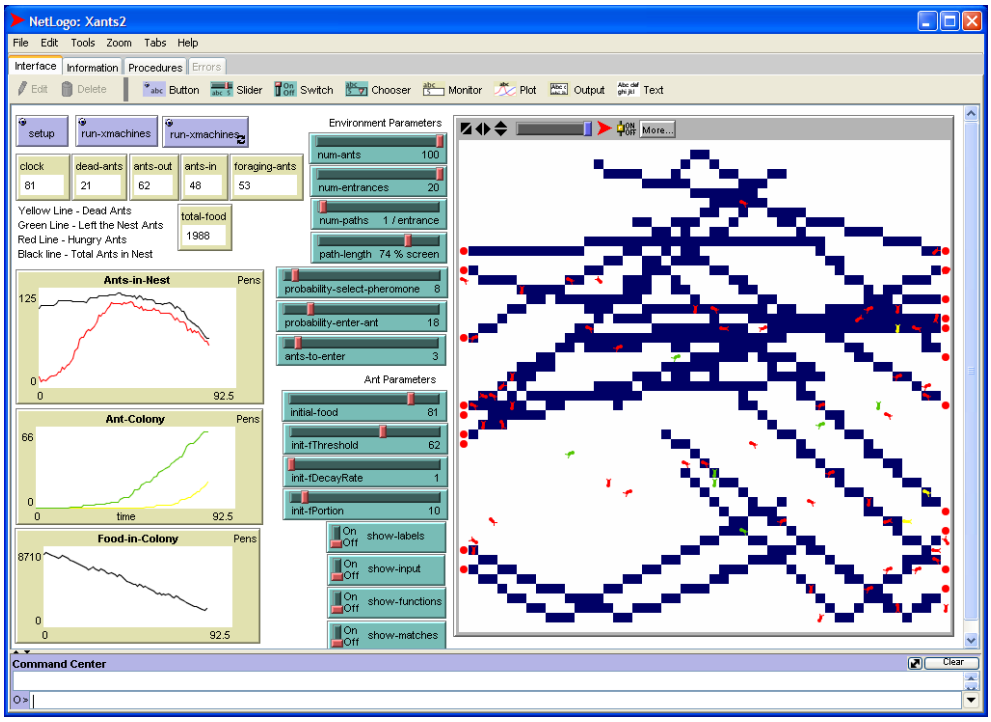

Fig. 3. A screen-shot of the simulator environment in NetLogo.

Finally, the simulator includes a graphical interface through which experiment parameters can be set and values describing the state of the modelled nest can be monitored. Figure 3 shows a screen-shot of the implemented application. The animation of the nest is displayed on the window on the right side of the screen-shot, where the lines represent pheromone paths. On the left hand side we can use the controls to set values for various parameters affecting the course of the simulation. Indicatively, some of the parameters we have used are the pheromone evaporation rate, the probability of an outside ant carrying food to enter the nest, the probability of an ant to choose a particular pheromone trail, the number of entrances and initial trails etc. (note that although the pheromone laying behaviour has not been explicitly modelled in $OPERAS_{XC}$, pheromone trails and foraging ants entering the nest have been included in the NetLogo simulation). An extensive experimentation with the values of the above parameters will be able to provide valuable feedback on the ways they affect the cooperation of the ants and the life of the colony.

6. Conclusions and Further Work

In this paper we have presented how the $OPERAS$ framework can be instantiated into a concrete formal method using X-machines and population P systems ($OPERAS_{XP}$), which allows the formal modelling of biologically inspired multi-agent systems, facilitating at the same time the simulation of the system in NetLogo. We demonstrated the applicability and intuitiveness of the proposed framework by formally modelling the in-nest behaviour of a Pharaoh ant colony. Through the animation, it is possible for the developers to informally verify that the model corresponds to the actual system under development. The animation can also be used to demonstrate the model to the end-users (with no technical background) allowing them to identify any misconceptions regarding the initial requirements offering informal validation of the expected behaviour.

Moreover, the developed model provides a mathematical modelling formalism for the system, which in turn allows X-machine specification to be model checked. Model checking of X-machine models is supported by $\mathcal{X}m$CTL, which enables the designer to verify the developed model against $\mathcal{X}m$CTL temporal logic formulas that express the properties that the system should have [5]. Following the implementation of the system, a complete test method also exists, which can generate all necessary test cases. This allows the use of a formal testing strategy to test the implementation and prove its correctness with respect to the X-machine model if certain assumptions in the implementation hold [6] and it is thus possible to assure that all desired properties of the system under development hold in the final product. Our approach also employs a simulation, allowing the informal verification of complex systems in cases where formal verification is impossible or impractical.

In terms of the animation, we considered the case of NetLogo as a suitable platform for the development of prototype simulations and in-silico experimentation due to the friendly environment and tools it provides. The NetLogo implementation was carried out in a rather straightforward way, bearing in mind the formal X-machine models of the individual ants and the reconfiguration rules. The formal model has driven the creation of the simulation, by mapping the XM functions and the PPS reconfiguration rules to NetLogo functions.

The implementation process itself has facilitated better understanding of the model and has helped us correct some ambiguities that existed in the original model. The easiness with which experiments can be parametrised can allow researchers to make preliminary observations on fundamental issues of self-organisation and emergence that could be used to complement in-vitro experiments. The NetLogo implementation revealed a framework under which similar simulations could be achieved in other domains of biology-inspired multi-agent systems.

Future work concerning the NetLogo animator involves the automatic translation of XMDL specifications in NetLogo code. Some steps have already been taken towards this direction [12]. Given the current structure of the X-machine meta-interpreter and the existing automatic XMDL to Prolog translation tools, this task is considered to be more than feasible.

Acknowledgement. We would like to express our special thanks to Prof. Tudor Bălănescu for his valuable contribution to the conception of the communicating X-machines model. His initial work and the discussions we had ten years ago motivated our work since then.

References

[1] BĂLĂNESCU T., COWLING A.J., GEORGESCU H., GHEORGHE M., HOLCOMBE M., VERTAN C., *Communicating Stream X-Machine Systems Are no more than X-Machines*, Journal of Universal Computer Science, **5**, 9, 1999, pp. 494–507.

[2] BERNARDINI F., GHEORGHE M., *Population P Systems*, Journal of Universal Computer Science, **10**, 5, 2004, pp. 509–539.

[3] EILENBERG S., *Automata, Languages and Machines*, Academic Press, 1974.

[4] ELEFTHERAKIS G., *Formal Verification of X-Machine Models: Towards Formal Development of Computer-based Systems*, PhD thesis, Department of Computer Science, University of Sheffield, 2003.

[5] ELEFTHERAKIS G., KEFALAS P., *Formal Verification of Generalised State Machines*, 12th Panhellenic Conference in Informatics, PCI'08, 2008; accepted.

[6] HOLCOMBE M., IPATE F., *Correct Systems: Building a Business Process Solution*, Springer, London, 1998.

[7] KAPETI E., KEFALAS P., *A Design Language and Tool for X-Machines Specification.* In: D.I. Fotiadis, S.D. Spyropoulos (Eds.) Advances in Informatics, World Scientific Publishing Company, 2000, pp. 134–145.

[8] KEFALAS P., ELEFTHERAKIS G., KEHRIS E., *Communicating X-Machines: A Practical Approach for Formal and Modular Specification of Large Systems*, Journal of Information and Software Technology, **45**, 5, 2003, pp. 269–280.

[9] KEFALAS P., STAMATOPOULOU I., GHEORGHE M., *A Formal Modelling Framework for Developing Multi-agent Systems with Dynamic Structure and Behaviour.* In: M. Pechoucek, P. Petta et al. (Eds.) Proceedings of the 4th Intern. Central and Eastern European Conf. on Multi-Agent Systems (CEEMAS'05), Lecture Notes in artificial Intelligence, 3690, pp. 122–131.

[10] PĂUN Gh., *Computing with Membranes*, Journal of Computer and System Sciences, **61**, 1, 2000, pp. 108–143.

[11] PĂUN Gh., *Membrane Computing. An Introduction*, Springer–Verlag, Berlin, 2002.

[12] PENEVA K., KEFALAS P., *Animating Formal Models of Biologically-Inspired Multi-Agent Systems.* In: Proceedings of the Balkan Conference in Informatics (BCI'05), 2005.

[13] STAMATOPOULOU I., KEFALAS P., GHEORGHE M., *Specification of Reconfigurable MAS: A Hybrid Formal Approach.* In: G. Antoniou, G. Potamias et al. (Eds.) Proceedings of the 4th Hellenic Conference on AI (SETN'06), Lecture Notes in Artificial Intelligence, 3955, 2006, pp. 592–595.

[14] STAMATOPOULOU I., KEFALAS P., GHEORGHE M., *Modelling the Dynamic Structure of Biological State-based Systems*, BioSystems, **87**, 2–3, 2007, pp. 142–147.

[15] STAMATOPOULOU I., KEFALAS P., GHEORGHE M., *OPERAS: a Formal Framework for Multi-Agent Systems and its Application to Swarm-based Systems*. In: A. Artikis, G. O'Hare et al. (Eds.) Proceedings of the 8th International Workshop on Engineering Societies in the Agents World (ESAW'07), 2007, pp. 208–223.

[16] STAMATOPOULOU I., KEFALAS P., GHEORGHE M., *OPERAS$_{CC}$: An Instance of a Formal Framework for MAS Modelling based on Population P Systems*. In: G. Eleftherakis, P. Kefalas, G. Păun (Eds.) Proceedings of the 8th Workshop on Membrane Computing (WMC'07), 2007, 551–566.

[17] WILENSKY U., *Netlogo*, `http://ccl.northwestern.edu/netlogo`. Center for Connected Learning and Computer-based Modelling. Northwestern University, Evanston, IL, 1999.