


An Agent Based Approach of Collective Foraging

View metadata, citation and similar papers at core.ac.uk

brought to you by  CORE

provided by idUS. Depósito de Investigación Universidad de Sevilla

Pérez Jiménez

Abstract. In this paper the behaviour of a bee colony is modeled as a society of communicating agents acting in parallel and synchronizing their behaviour. Two computational models for defining the agents behaviour are introduced and compared and tools developed for these models are briefly illustrated.

1 Introduction

An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives [11]. There are two fundamental concepts associated with any dynamic or reactive system, such as an agent, that is situated in and reacting with some environment [9]:

- the environment itself must be defined in some precise, mathematical way, something which involves identifying the important aspects of the environment and the way in which they may change in accordance with the activities of the agent;
- the agent will be responding to environmental changes by changing its basic parameters and possibly affecting the environment as well. Thus, there are two ways in which the agent reacts, i.e. it undergoes internal changes and it produces outputs that affect the environment.

Agents, as highly dynamic systems, are concerned with three essential factors:

- a set of appropriate environmental stimuli or inputs;
- a set of internal states of the agent;
- a rule that relates the two above and determines what the agent state will change to if a particular input arrives while the agent is in a particular state.

One of the challenges that emerge in intelligent agent engineering is to develop agent models and agent implementations that are *correct*. According to [9], the criteria for *correctness* are:

- the initial agent model should match with the requirements;
- the agent model should satisfy any necessary properties in order to meet its design objectives;
- the implementation should pass all tests constructed using a complete functional test generation method.

All the above criteria are closely related to three stages of agent system development, i.e. *modelling, verification and testing*.

In agent oriented engineering, there have been several attempts to use formal methods, each one focusing on different aspects of agent systems development. One of them was to formalize *PRS* (Procedural Reasoning System), a variant of the *BDI* architecture [15] with the use of *Z*, in order to understand the architecture in a better way, to be able to move to the implementation through refinement of the specification and to be able to develop proof theories for the architecture [10]. Trying to capture the dynamics of an agent system, [16] viewed an agent as a situated automaton that generates a mapping from inputs to outputs, mediated by its internal state. [6] developed the *DESIRE* framework, which focuses on the specification of the dynamics of the reasoning and acting behaviour of multi-agent systems. In an attempt to verify whether properties of agent models are true, work has been done on model checking of multi-agent systems with re-use of existing technology and tools [5], [14]. Towards implementation of agent systems, [2] focused on program generation of reactive systems through a formal transformation process. Finally, in a less formal approach, extensions to *UML* to accommodate the distinctive requirements of agents (*AUML*) were proposed [17].

A specific class of agents, namely cooperative ones, have been used to modelling the collective foraging behaviour of a colony of honey-bees [8]. This class of agents is fully compatible with the rules used by foraging honey-bees that include specifications for [18]:

- travelling from the nest to the source;
- searching for the source;
- collecting nectar from the source;
- travelling back to the nest;
- transmitting the information about the source - the dancing of the returning bee;
- the reaction of a bee in the nest to the dancing of a nest mate.

In this paper, we intend to show how simple agents motivated from biology can be modelled as P systems [13] and X-machines [9]. Such modelling will facilitate both verification and testing of an agent model, since appropriate strategies for model checking and testing are already developed around these methods. In addition, modular construction of agent models is feasible since these models are provided with communicating features, which allow simple models to interact. Finally, tools developed for these models are briefly presented in order to demonstrate the practicality of the approach. The two models present many similarities, but also important differences making them complementary approaches which is an important characteristic of the models developed in the area of biochemical systems [7].

2 P systems and stream X-machines. Basic definitions

Definition 1. A P system $\Pi = (V, T, \mu, M_1, \dots, M_m, R_1, \dots, R_m)$ is a construct, where:

- V is an alphabet; its elements are called objects;
- $T \subseteq V$ is the output alphabet;
- μ is a membrane structure (a rooted tree where the nodes are called membranes and the root is called skin) consisting of m membranes, with the membranes and the regions labeled in a one to one manner with elements $1, \dots, m$;
- M_1, \dots, M_m are multisets over V associated with the regions $1, 2, \dots, m$;
- R_1, \dots, R_m are finite sets of rewriting (evolution) rules of the form $a \longrightarrow x$ where a is a word over V and x is a word over $\{a_{here}, a_{in}, a_{out} \mid a \in V\}$.

Usually the membrane structure is materialized by matching pairs of parentheses. The multisets M_i and the rule sets R_i are associated with the regions of μ . When a rule $a \longrightarrow x$ is used in a region then the objects designed by a are removed, and the objects designed by x are sent to the regions indicated by tar . When $tar = here$, the object is kept in the same region; when $tar = out$, the object leaves the current region and goes into the outer one; when $tar = in$, the object is sent to one of the directly included regions, if any exists, otherwise the rule can not be applied. The target indication *here* is in general omitted. A computation is defined as follows: the process starts with the multisets present in the initial configuration and proceeds iteratively by applying in parallel the rules in each region to all multisets that can be rewritten. The result (the number of objects) is collected outside of the system at the end of the halting computation. The language generated by a system Π is denoted by $L(\Pi)$ and consists of all numbers that are computed during halting computations. Other ingredients are also considered alongside of such a definition: priority rules among the components of the sets R_i , membrane polarization or the permeability (thickness) of a membrane etc., [13]. Also the symbol-objects with no internal structure may be replaced by string-objects and in this case we deal with usual languages over a given alphabet.

Definition 2. A stream X-machine $X = (\Sigma, \Gamma, Q, M, \Phi, F, I, T, m_0)$ is a system, where:

- Σ and Γ are finite sets called the input and the output alphabets, respectively;
- Q is the finite set of states;
- M is a (possibly infinite) set of memory symbols;
- Φ is a set of basic partial functions of the form $f : \Sigma \times M \longrightarrow M \times \Gamma^*$;
- F is the next state function $F : Q \times \Phi \longrightarrow 2^Q$;
- I and T are the sets of initial and final states;
- m_0 is the initial memory value.

A computation in X is defined as follows: the process starts with the initial configuration (m_0, q_0, s, ϵ) , where $q_0 \in I$, $s \in \Sigma^*$, ϵ denotes the empty output, and proceeds iteratively by processing the current input symbol - the symbol on the left of the input string - and the current memory value by a function f emerging from the current state q , producing an output symbol going to the right of the output string and updating the memory value; the next state is one belonging to $F(q, f)$. The machine will stop when arrives in a final state and has processed all the input symbols of s . The output string obtained in this state is the result of the computation. The set of all output strings obtained by using all the possible computations in X starting with an input set Imp is denoted by $L_X(Imp)$.

Comparisons involving these models based on their computational power, when they both are dealing with strings, has been investigated [1], and ways of combining them into a hybrid model called Eilenberg P system have been proposed [4]. For such a model has been shown that NP-complete problems like SAT can be solved in linear time [4].

3 Two computational models of collective foraging

In this section an agent based approach using both P systems and X-machines will be used in order to specify the collective foraging behaviour of a colony of honey-bees.

The P system model has the following elements organized on three layers:

- the environment (M_1) containing the nectar source;
- the nest (M_2);
- the bees (M_3 to M_m).

M_1 will contain information about the amount of nectar carried by a bee, its current position, and information about the source. M_2 will have for each bee in the hive, the amount of nectar carried, its current position, a memory value identifying the nectar source position as well as an identification of each bee. M_i , $1 \leq i \leq m$ will give the position of a bee, the amount of nectar carried by a bee and a position of the source as a memory value; when some nectar will be transferred to another bee, the position and amount to be transferred will be also specified.

The membrane structure is: $\mu = [{}_1[{}_2[{}_{i_1}]_{i_1} \cdots [{}_{i_p}]_{i_p}]_{i_2} [{}_{i_{p+1}}]_{i_{p+1}} \cdots [{}_{i_{m-2}}]_{i_{m-2}}]_1$.
The following rules are applied:

- 1. $(nectar, p, m) \longrightarrow (nectar, p', m')$, where p, p' are positions, $nectar$ is an arbitrary amount of nectar from a finite set of values and m is the memory content defining the position of the source; this rule mentions that a bee changes its position and may update its memory;
- 2. $(0, p_source, m) \longrightarrow (nectar_taken, p_source, m)$ - this rule describes what a bee is doing once she has arrived at the source: the current amount of nectar she carries is 0, the source position is p_source , the bee picks up the amount $nectar_taken$ and keeps the position;
- 3. $(nectar, p, m) \longrightarrow (nectar, p, m, i)_1$ - according to this rule a foraging bee i being in the environment communicates its load, position and identity to the environment (the element on the right hand side is sent to the environment, 1); similarly we have 3'. $(nectar, p, m) \longrightarrow (nectar, p, m, j)_2$ - the bee j communicates with the hive;
- 4. $(nectar, p, m, i) \longrightarrow (nectar, p, m, i)_2 (0, p, m, i)_2^k$ according to this rule, from the environment it passes to the hive the nectar load of the foraging bee i and k copies indicating the source position simulating the information that is passed through the waggle dance to k bees (the amount of nectar is not considered);
- 5. $(nectar, p_1, m_1, i)(nectar, p_2, m_2, j) \rightarrow (nectar'_1, p_1, m_1, i)_{(1)}(nectar'_2, p_2, m_2, j)$ - this rule shows that the bees i and j exchange nectar; the first object could go to the environment or rest in the hive (the notation (1) means either 1 i.e. the environment or nothing which means the objects remains in the hive);
- 6. $(nectar, p, m, i) \longrightarrow (nectar, p, m)_i$ - this rule shows that the bee i (re)starts foraging after nectar has been given to honey-bees in the hive;
- 7. $((*, p_1, source_info, i), (nectar_2, p_2, m, j)) \longrightarrow (nectar_2, p_2, new_m)_j$ - according to this rule the bee j who attends the dance will refresh her memory with the position of the source; depending on the distance between the two bees i and j (the difference between p_1 and p_2), a transmission error may occur [18].

The rules 1, 2, 3 or 3' are in the sets R_i , $3 \leq i \leq m$, whereas the rules 5, 7 are only in R_2 and 4, 6 in R_1 . The sets R_i , $i \geq 1$ formally define the six requirements stated in [18]. In the next section we will show how these rules are codified in a software tool which helps to simulate the behaviour of this system. V contains elements $(nectar, p, m)$ or $(nectar, p, m, i)$ with $0 \leq nectar \leq Nectar_max$, $0 \leq p \leq Position_max$, $0 \leq memory \leq Postion_max$, $3 \leq i \leq m$; where $Nectar_max$ and $Position_max$ are two positive integers.

The next model will approach the foraging problem from a different perspective. In this case instead of modeling the whole problem, we will approach different aspects and model them as separate X-machines. At the end we will have a set of such machines. In [12] it is proposed a way of aggregating these components by providing a protocol for communication among them. We will illustrate here only the X-machine which models the dancing behaviour.

The X-machine X will have the following elements:

- $\Sigma = \{fbee, space, nest, source\}$;
- $\Gamma = \{“dancing”, “flying out”, “flying in”, “source found”, “keep flying”\}$;
- $Q = \{in\ the\ nest, out\ of\ nest\}$;
- $M = \{(bee_pos, source_pos) \mid bee_pos, source_pos \in Fin\}$, where Fin is a finite set of integer values $\{0, \dots, Position_max\}$;
- Φ contains
 - $dancing(fbee, (bee_pos, source_pos)) = ((bee_pos, source_pos), “dancing”)$;
 - $fly_out(space, (bee_pos, source_pos)) = ((bee_pos', source_pos), “flying\ out”)$;
 - $fly_in(nest, (bee_pos, source_pos)) = ((bee_pos', source_pos), “flying\ in”)$;
 - $find_source(source, (bee_pos, source_pos)) = ((source_pos, source_pos), “source\ found”)$;
 - $keep_fly_out(space, (bee_pos, source_pos)) = ((bee_pos', source_pos), “keep\ flying”)$.
- $F(in\ the\ nest, dancing) = in\ the\ nest$; $F(in\ the\ nest, fly_out) = out\ of\ nest$;
- $F(out\ of\ nest, fly_in) = in\ the\ nest$;
- $F(out\ of\ nest, find_source) = out\ of\ nest$;
- $F(out\ of\ nest, keep_fly_out) = out\ of\ nest$;
- $I = \{in\ the\ nest\}$; $F = Q$.

In both systems either their rules (in the first case) or the X-machine components (in the second case) run in parallel approaching the honey-bees behaviour as cooperative agents.

4 Tools

In this section we will show how the formal models defined in the previous section might be transformed into an executable code such as to be able to simulate the behaviour of the defined system.

A P system simulator has been implemented; it allows to input a P system specification and then to simulate its nondeterministic and highly parallel behaviour [3].

The MzScheme code for the P system model defined in the previous section, is presented below (only the rules for the environment are provided):

```
(define A (vector (n1 p1 m1) ... (nk pk mk) (n1 p1 m1 3) ... (n1 p1 m1 m)
                 ...
                 (nk pk mk 3) ... (nk pk mk m)))
;the alphabet codifying the system objects
(define MS ((1 2) (1 3) (1 4) ... (1 p)
           (2 p+1) ... (2 m)); the membrane structure
(define objects
  (vector ((n1 p1 m1 3) ... (nh ph mh j)))
; initial configuration
(define rules
  (vector (((n p m i)->(n p m i) 2)((n p m i) 2) ... ((n p m i) 2))
          ((n p m i)->(n p m i)) ; rules corresponding to 4 and 6
  )
)
```

The X-machine model is supported by a mark-up language called XMDL [12]. An XMDL fragment code of the X-machine specification of the model defined in the previous section is given below:

```
#input = {fbee, space, nest, source}.
#output = {"dancing", "flying out", "flying in", "source found",
           "keep flying"}.
#memory = {(b,s) | 0<=b<=Position, 0<=s<=Position}.
#state = {in_the_nest, out_of_nest}.
#fun dancing(?inp,(?b_p,?s_p)) =
    if ?inp==fbee then ((?b_p,?s_p),"dancing").
#fun fly_out(?space,(?b_p,?s_p)) =
    if next(?b_p,?s_p,?b_p',?s_p') then ((?b_p',?s_p'),"flying out").
```

From the above sample code we may notice the fair similarity between their formats and the formalisms used in these models. Both tools are able to animate the specified systems and XMDL is also providing a sort of formal analysis of the system by generating test sets and supporting model checking analysis.

5 Conclusions

The work reported in this paper refers to two computational models that are proposed as cooperative agent paradigms suitable to modelling the behaviour of social insects, in particular honey-bee colonies. The models are parallel distributed paradigms with specific communication mechanisms and are supported by software tools able to animate the systems specified in these frameworks.

These approaches have some similarities: they are computational devices with components running in a fully parallel manner and acting as a society of agents. For both models the design is flexible and reusable as the whole system may be built from individual components (multisets and rules in the membrane approach and component X-machines in the second case) that are assembled and enriched with the communication aspects. There are also some important differences between the two approaches. In the case of the membrane systems the outputs are defined only at the end of the computation and in some specified components; the inputs are defined only in some variants, the memory is implicitly defined as being the set of symbols associated to each component. For X-machines the inputs and the outputs are explicitly associated to every basic function and a memory element is part of any computation.

The theoretical study of these models has been developed independently and somehow on different aspects, computational power for P systems [13] and testing issues for X-machines [9]. Consequently some more work is expected in order to combine these approaches [4]. On the practical side of modeling the behaviour of different social insects, more powerful and flexible tools are requested to simulate and animate various aspects related to the evolution of these colonies and to capture a broader range of system specifications.

References

1. J. Aguado, T. Bălănescu, T. Cowling, M. Gheorghe, M. Holcombe, F. Ipate, P systems with replicated rewriting and stream X-machines (Eilenberg machines), *Fundamenta Informaticae* 49(2001), 1–17.
2. A. Attoui, A. Hasbani, Reactive systems developing by formal specification transformations. In *Proceedings of the 8th International Workshop on Database and Expert Systems Applications* (DEXA 97), 1997, 339–344.
3. D. Balbotín Noval, M. J. Pérez Jiménez, F. Sancho Caparrini, A MzScheme Implementation of Transition P Systems. In *Membrane Computing* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597, Springer-Verlag, 2003, 58–73.
4. T. Bălănescu, M. Gheorghe, M. Holcombe, F. Ipate, Eilenberg P systems. In *Membrane computing* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597, Springer-Verlag, 2003, 43–57.
5. M. Benerecetti, F. Giunchiglia, L. Serafini, A model checking algorithm for multi-agent systems. In *Intelligent Agents V* (J. P. Muller, M. P. Singh, A. S. Rao, eds.), LNAI 1555, Springer-Verlag, 1999, 163–176.
6. F. Brazier, B. Dunin-Keplicz, N. Jennings, J. Treur, Formal specification of multi-agent systems: a real-world case. In *Proceedings of International Conference on Multi-Agent Systems* (ICMAS'95), MIT Press, 1995, 25–32.
7. L. Clark, R. Paton, Towards computational models of chemotaxis in *Escherichia coli*. In *Information Processing in Cells and Tissues* (M. Holcombe, R. Paton, eds.), Plenum Press, 1998, 39–46.
8. M. Gheorghe, M. Holcombe, P. Kefalos, Computational models of collective foraging, *BioSystems*, 61(2001), 133–141.
9. M. Holcombe, F. Ipate, *Correct systems: Building a Business Process Solution*, Springer Verlag, London, 1998.
10. M. d’Inverno, D. Kinny, M. Luck, M. Wooldridge, A formal specification of dMARS. In *Intelligent Agents IV* (M.P.Singh, A.Rao, M.J.Wooldridge, eds.), LNAI 1365, Springer-Verlag, 1998, 155–176.
11. N. R. Jennings, On agent-based software engineering, *Artificial Intelligence*, 117(2000), 277–296.
12. E. Kapeti, P. Kefalos, A design language and tool for X-machines specification. In *Advances in Informatics* (D. I. Fotiadis, S. D. Nikolopoulos, eds.), World Scientific Publishing Company, 2001, 134–145.
13. Gh. Păun, *Membrane Computing. An Introduction*, Springer, Berlin, 2002.
14. A. S. Rao, M. P. Georgeff, A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (IJCAI'93) (R. Bajcsy, ed.), 1993, 318–324.
15. A. S. Rao, M. P. Georgeff, BDI Agents: from theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems* (ICMAS95), 1995, 312–319.
16. S. R. Rosenschein, L. P. Kaelbling, A situated view of representation and control, *Artificial Intelligence*, 73(1995), 149–173.
17. J. Odell, H. V. D. Parunak, B. Bauer, Extending UML for agents. In *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, 2000.
18. H. de Vries, J. C. Biesmeijer, Modelling collective foraging by means of individual behaviour rules in honey-bees, *Behavioral Ecology and Sociobiology*, 44(1998), 109–124.