Computational models of collective foraging

Marian Gheorghe[1], Mike Holcombe[1], Petros Kefalas[2]
[1]Department of Computer Science, University of Sheffield, Sheffield, UK
[2] Department of Computer Science, City College, Thessaloniki, Greece

Abstract. In this paper the behaviour of a bee colony is modelled as society of communicating agents acting in parallel and synchronising their behaviour. Two computational approaches for defining the agents behaviour are introduced and compared. Their common features as well as the complementary aspects making them suitable for merging together into a more complex model.

INTRODUCTION

Computational models based on (collaborative) agents behaviour have been studied [5]. Three main characteristics of any agent have been identified [16]: autonomy, learning and cooperation. Autonomy refers to the principle that agents can operate on their own, hence they have their individual states and goals. Learning defines the capacity of the agents to react and interact smartly with the environment. Cooperation with other agents is paramount being the raison d'être for having many agents instead of only one. The combination of these characteristics gives birth to various types of agents: collaborative agents, collaborative learning agents, interface agents and smart agents. The first class identified corresponds to those agents having mostly autonomy and being cooperative. Both theoretical aspects pointing out the logic underpinning the collaborative agents as well as the practical issues showing how they are applied to large-scale applications have been investigated [12]. A specific model of multi agent paradigm called *cooperating distributed grammar systems*, has been devised [2], investigated and developed [11]. Agents have been also described and investigated using logic based notations [15], and specific agent-based communication languages such as KQML [3]. Another approach is a more operational approach using generalised state machines, to demonstrate how the behaviour of agents can be captured, in a natural way [8]. The capabilities of different levels of agent can be conveniently described by using the parameters such as memory structure, basic function and other components.

The cooperative agents may be a good candidate for modelling the collective foraging behaviour of a colony of honey-bees as it is fully compatible with the rules used by foraging honey-bees that include specifications for [14]: (1) travelling from the nest to the source, (2) searching for the source, (3) collecting nectar from the source, (4) travelling back to the nest, (5) transmitting the information about the source – the dancing of the returning bee -, (6) the reaction of a bee in the nest to the dancing of a nest mate. It is essential to the model to provide a mechanism for representing the bee memory [14].

In this paper we introduce an agent-based approach by discussing two computational models with many similarities but with important differences, in order to get the benefits from both and to propose some ways of combining or complementing them. The usefulness of complementary models in the area of biochemical systems has been already pointed out [1]. The first approach is based on a new rapidly growing research area called *membrane* or *P systems* [10] whereas the second uses a relatively new variant of the stream X-machines [6], namely *communicating stream X-machines* [9]. The first model has been already used for describing various living systems [13] and

the second has proved to be suitable for defining metabolic pathways [5] or the behaviour of collaborating ants [8]. In both approaches we deal with a set of agents each with a variety of attributes and capabilities that cooperate with each other to achieve a number of tasks. Each agent can carry out a number of actions, while a task may require several agents to operate in parallel or one after the other until the task is completed. Each agent is specified by a set of rules, these rules involve conditions determined by the environment, the inputs to the system and by the state of the system, an internal memory device.

THE MEMBRANE MODEL

A membrane system is a construct:
$\Delta = (A, T, \mu, M_1,\ldots,M_n, R_1,\ldots R_n)$
where

- A is a set of objects and $T \subseteq A$;
- μ is a membrane structure (it can be changed throughout a computation) that defines a hierarchical structure of nested regions;
- $M_1,\ldots,M_n$ are multisets associated to the n regions of the membrane system;
- $R_1,\ldots R_n$ are rules associated to the n regions.

The multisets may be replaced by sets of words over A and the rules may be applied according to some priorities (expressed as order relationship). The rules have the format $u \rightarrow v$, where u is a word over A and v is a word over $\{a_{here}, a_{out}, a_{tar} | \ a \in A, 1 \leq tar \leq n\}$. The rules of each R_i are applied to the elements of M_i in a totally parallel way in every region i. The symbols having the index *here* will be kept in the current region (subsequently this index will be omitted), those with index *out* will be sent to the outer region containing the current region and those with index *tar* will be sent to the region identified by *tar* and which should be immediately included in the current region. When no rule may be applied the computation stops and the results is read from a specified region or from outside.

Example. Let us consider the following simple membrane system Δ having the components (Fig.1):
$A=T=\{a,b,c\};$
$M_1 = \{a\}, M_2 = M_3 = M_4 = \varnothing;$
$R_1 = \{a \rightarrow aa, a \rightarrow c, a \rightarrow b_2, a \rightarrow b_3\},$
$R_2 = \{b \rightarrow c\},$
$R_3 = \{b \rightarrow c, b \rightarrow c_4\},$
$R_4 = \varnothing.$
The multisets contain in all the four regions the same occurrences of c. If L_i denotes the multiset of the region i then $L_i = \{ c^n \mid n \geq 0\}$.

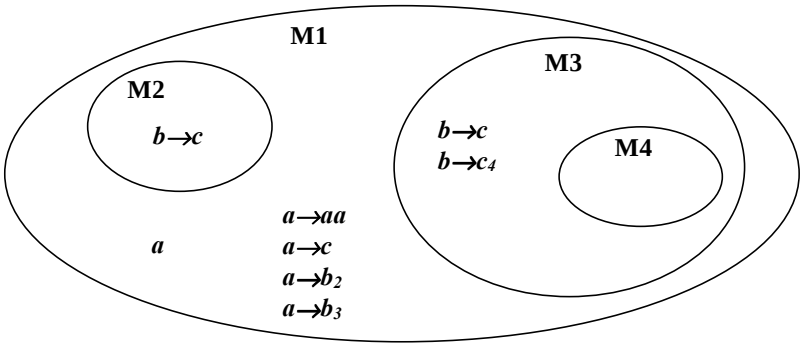


Fig. 1. An example of a simple membrane system.

The agent-based approach to model the collective foraging behaviour of a colony of honey-bees has the following elements organized on three layers: (1) the environment (M_1) containing the nectar source, (2) the nest (M_2) and (3) the bees (M_3 to M_n). 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, $3 \le i \le n$, will give the position of a bee, the amount of nectar carried by a bee and the 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 rules are shown in Table1.

1	*(nectar,p,m)* $\rightarrow$ *(nectar,p',m')* , *where p, p'* $\in$ *Set_of_positions, nectar* is an arbitrary amount of nectar and *m* is the memory content defining the position of the source - this is for changing the position of a bee, with or without a given amount of nectar and knowing more or less exactly the position of the source – this could be improved with a more accurate *m'* -;
2	*(0,p_source,m)* $\rightarrow$ *(nectar_taken, p_source,m)* – this is to describe that a bee arrives at the source (the amount of nectar she has now is *0* and the position of the source is *p_source*) and picks up the amount *nectar_taken* and keeps the position;
3	*(nectar,p,m)* $\rightarrow$ $(nectar,p,m,i)_1$ – according to this rule a foraging bee, *i,* being in the environment "communicates" its load, position and identity; *(nectar,p,m)* $\rightarrow$ $(nectar,p,m,j)_2$ –similar for bee, *j*, in the hive;
4	*(nectar,p,m,i)* $\rightarrow$ $(nectar,p,m,i)_2(*,p,m,i)^k_2$ – according to this rule from environment pass to hive the nectar load of foraging bee *i* and *k* copies indicating the source position – waggle dance;
5	$((nectar,p_1,m_1,i),(nectar_2,p_2,m_2,j)) \rightarrow (nectar'_1,p_1,m_1,i)_{(1)}\ (nectar'_2,p_2,m_2)_j$ – this rule shows that the bees *i* and *j* exchange nectar; the first object could go to the environment or rest in the hive: (1) means 1 – environment – or – remain in nest
6	$(nectar,p,m,i) \rightarrow (nectar,p,mi)_i$ – this is to model that the foraging bee *i*, restarts nectar searching
7	$((*,p_1,source\_info,i),(nectar_2,p_2,m,j)) \rightarrow (nectar_2,p_2,new\_m)_j$ – according to this rule the bee who attends the dance will refresh the 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.

Table 1. The rules for the collective foraging behaviour of a membrane system.

The rules 1, 2, 3, or 1,3' are in the sets R_i, $3 \leq i \leq n$, whereas the rules *5,7* are only in R_2 and *4,6* in R_1. Figure 2 shows the membrane system of the honey-bee colony.

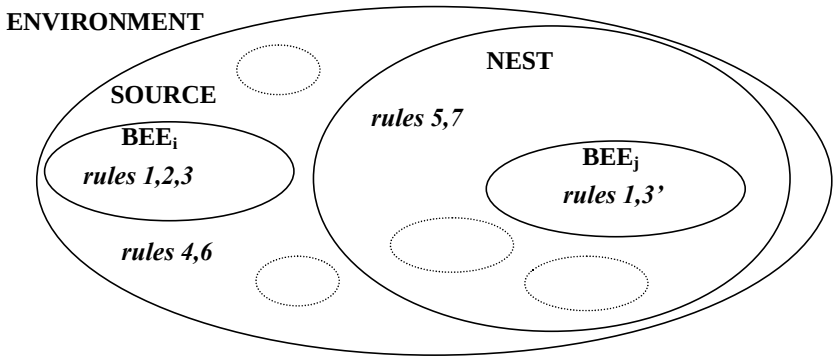


Fig. 2. The membrane system that models for the collective foraging behaviour of honey-bees.

The sets of rules R_i formally define the six requirements stated in [14]. In order to get a computational model which mimics enough accurate the behaviour of collective foraging in a nest some values of the parameters concerning the transmission accuracy, search ability or the abandoning probability should be considered. With such values some simulations may be developed with the model.

THE COMMUNICATING STREAM X-MACHINE MODEL

A X-machine is a general computational machine that resembles a Finite State Machine (FSM) but with two significant differences: (a) there is memory attached to the machine, and (b) the transitions are not labelled with simple inputs but with functions that operate on inputs and memory values (Fig. 3). These differences allow the X-machines to be more expressive and flexible than the FSM. The machine, depending on the current state of control and the current values of the memory, consumes an input symbol from the input stream and determines the next state, the new memory state and the output symbol, which will be part of the output stream. The formal definition of a deterministic stream X-machine is an 8-tuple [6]:
$SXM = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$, where:

- Σ, Γ is the input and output finite alphabet respectively;
- Q is the finite set of states;
- M is the (possibly) infinite set called memory;
- Φ is the type of the *SXM* machine, a finite set of partial functions φ that map an input and a memory state to an output and a new memory state:
$$\varphi: \Sigma \times M \rightarrow \Gamma \times M$$
- F is the next state partial function that given a state and a function from the type Φ, denotes the next state. F is often described as a transition state diagram:
$$F: Q \times \Phi \rightarrow Q$$
- q_0 and m_0 are the initial state and memory respectively;

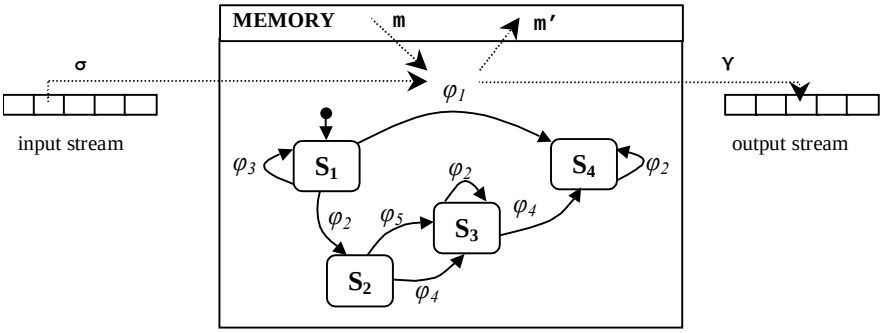


Fig. 3. An abstract example of a X-machine: φ_i - functions operating on inputs and memory, S_i - states. The general formal of functions is: $\varphi(\sigma,m) = (\gamma,m')$

Agents, e.g. foraging bees, can be modeled as stand-alone (possibly complex) X-machines. However, an agent can be also viewed as a set of simpler X-machines considered as components, which model various different behaviours of the agent, e.g. the rules for collective foraging behaviour. These behaviours integrate in order to result in a situated agent with the desired overall robust performance. A methodology of building communicating X-machines from existing stand-alone X-machine is developed [9] so that agent modeling can be split down into two separate activities: (a) the modeling of the X-machine components and (b) the description of the communication between these components. The approach has several advantages for the designer who: (a) does not need to model a communicating system from scratch, (b) can re-use existing models, (c) can consider modeling and communication as two separate distinct activities, and (d) can use existing tools for both stand-alone and communicating X-machines [7].

A foraging bee can be modeled according to set of independent behaviours, which constitute the components of the overall agent. Table 2 shows some of the behaviours of a foraging bee modeled as simple X-machines, with an input set Σ and a memory tuple M. Each machine has different memory, inputs (percepts), and functions. Some states and functions were on purpose named differently to show the modularity of the approach. It is assumed that the bee perceives empty space to fly (*space*), the hive (*nest*), the source of nectar (*source*), an amount of nectar (*nectar*), three types of other bees, a foraging bee (*fbee*) and a receiving bee (*rbee*), and finally understands when it has lost its orientation (*lost*). The memory of each X-machine holds information on the bee, the source and the nest positions (*bee_pos, source_pos* and *nest_pos*), the amount of nectar carried (*nectar_amount*), and its status (*employed* or *unemployed*).

For example, consider the X-machine modeling the dancing behaviour. Its functions are defined as follows:
dancing(fbee, (bee_pos, source_pos)) → ("dancing", (bee_pos, source_pos))
fly_out(space, (bee_pos, source_pos)) → ("flying out", (bee_pos', source_pos))
fly_in(nest, (bee_pos, source_pos)) → ("flying in", (bee_pos', source_pos))
find_source(source, source_pos)) → ("source found", (source_pos, source_pos))
keep_fly_out(space,(bee_pos ,source_pos)) → ("keep flying",(bee_pos', source_pos))
where *bee_pos, bee_pos', source_pos* $\in$ *Set_of_positions*. The bee position can be calculated by some external function or some other X-machine.

Other behaviours can be modeled accordingly, if required, as well as the environment, the source and the nest could be considered as other X-machines (see Fig.6).

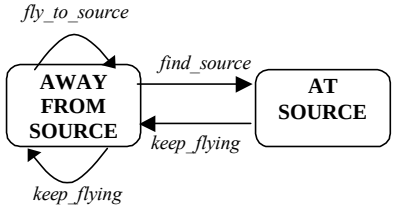
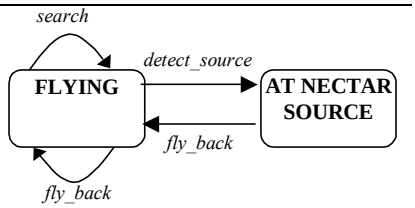
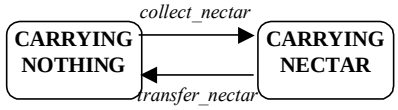
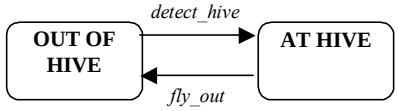
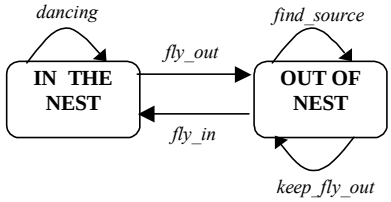
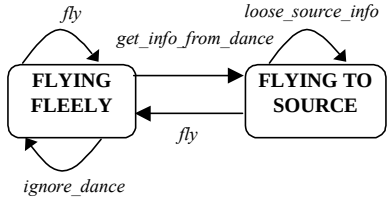
Behaviour	X-machine model
Traveling from nest to source: Σ = *{space, source}* *M = (bee_pos, source_pos)* q_0 = *at source*	
Searching for the source: Σ = *{space, source}* *M = (bee_pos, source_pos)* q_0 = *flying*	
Collecting nectar from the source: Σ = *{nectar, rbee}* *M = (nectar_amount)* q_0 = *carrying nothing*	
Traveling back to the nest: Σ = *{nest, space}* *M = (nest_pos)* q_0 = *at hive*	
Transmitting information about the source (dancing): Σ = *{fbee, space, nest, source}* *M = (bee_pos, source_pos)* q_0 = *in the nest*	
Reacting to the information transmitting by the dancing: Σ = *{space, lost, source_pos}* *M = (status, source_pos)* q_0 = *flying freely*	

Table 2. The behaviours of a foraging bee modeled separately as X-machine components.

If so annotated, the functions read input from a communicating stream instead of the standard input stream. Also, the functions may write to a communicating input stream of another X-machine. The normal output of the functions is not affected. The annotation used is the *solid circle* (IN port) and the *solid box* (OUT port) to indicate that input is read from another component and output is directed to another component respectively. For example, function φ in Fig.4 accepts its input from the model *x-m1* while writes its output to model *x-m2*. Multiple communications channel for a single X-machine may exist. The approach is practical, in the sense that the designer can separately model the components of an agent and then describe the way in which these components (behaviours) communicate. This allows a disciplined development of situated agents. Also, components can be re-used in other systems, since the only thing that needs to be changed is the communication part.

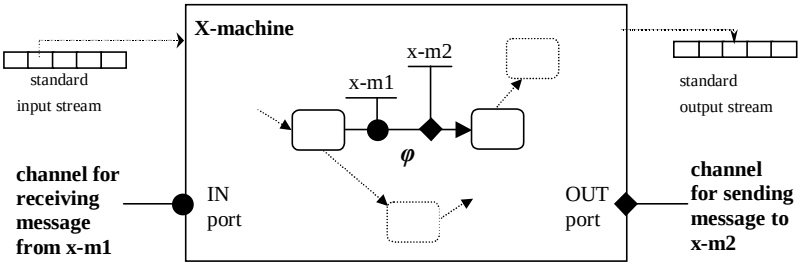


Fig. 4 An abstract example of a Communicating X-machine component.

Fig. 5 shows in detail how communication can be achieved directly by various honey-bees, e.g. an employed foraging bee sends the source position to another foraging bee through the dancing behaviour:

$dancing(fbee, (bee\_pos, source\_pos)) \rightarrow$

$(OUT_{x\text{-}m\ reacting}(source\_pos), (bee\_pos, source\_pos))$

while an unemployed foraging bee reads the source position by the function:

$get\_info\_from\_dance(IN_{x\text{-}m\ dancing}(source\_pos),(unemployed, nil)) \rightarrow$

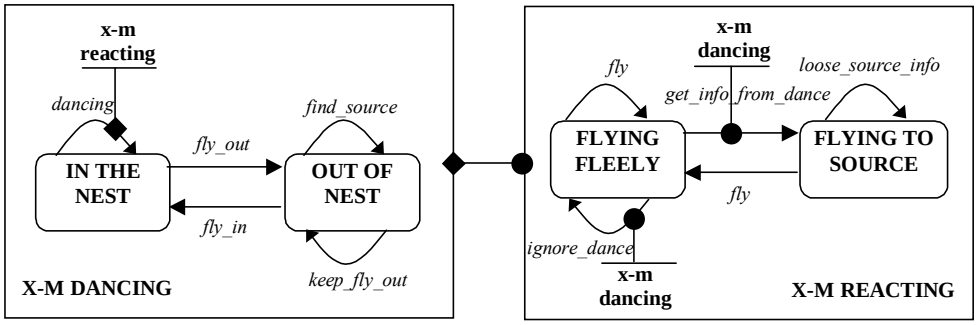
("getting source info", (employed, source_pos)).

If the foraging bee is currently employed, it just ignores the message:

$ignore\_dance(IN_{x\text{-}m\ dancing}(source\_pos),(employed, source\_pos)) \rightarrow$

("ignoring source info", (employed, source_pos)).

The same communication takes place when a foraging bee transfers the amount of nectar that is carrying to a receiving bee waiting at the hive.



Behaviour of a foraging employed bee **Behaviour of a foraging unemployed bee**

Fig. 5 An example of two communicating behaviours; an employed bee sends information about the source position to an unemployed bee.

Fig.6 shows part of the complete system (environment and colony), which is made up of component X-machines, which communicate via channels. Each machine works separately and concurrently in an asynchronous manner. Each machine can send a message through a communication channel (OUT port) will act as input to functions of another component (IN port). The figure implies that the environment as well as nest and the source can be modeled as X-machines too. The environment provides percepts to bees, so that $\Sigma = \{space, nest, source, nectar, rbee, fbee, lost\}$, which in turn consume these percepts by directing them to the appropriate behaviours. It is apparent that the receiving bee may have some, but not all the behaviours of a foraging bee. Due to the modularity of the approach, individual behaviours can be re-used to model any type of bee on demand.

The whole system works as follows: an employed bee accepts inputs from the environment, which cause transitions in the X-machine components that model its

individual behaviours. Having found a source and after collecting nectar (appropriate transitions have been performed in source and environment X-machines), the bee returns to hive and on the sight of another foraging bee performs the dancing which, as shown earlier, transmits the information of the source position. An unemployed bee accepts the input from its communication port and changes its status to employed. It can then perceive inputs from the environment and travels to the source. The whole process may then be repeated. In parallel, other bees can do the same in an asynchronous manner.

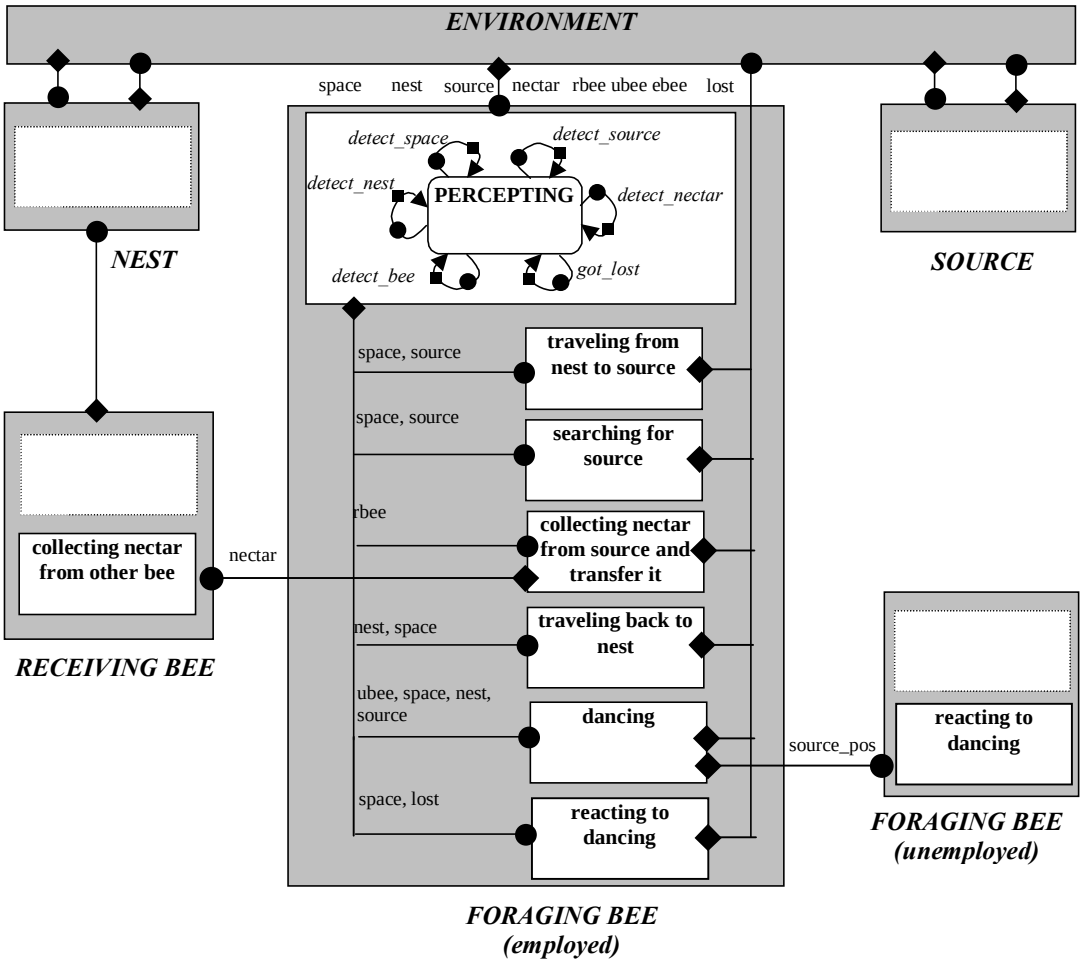


Fig. 4 An abstract example of a Communicating X-machine modeling the behaviour of the bee colony

We used communicating X-machines to formally define the requirements in [14]. It is then possible to develop various models of a single agent and multiple instances of that model which can communicate with the environment and between them using a communicating X-machine model.

CONCLUSIONS

An agent-based model using two computational approaches is introduced for describing the behaviour of a bee colony. In this model multiple agents act in parallel and cooperate like components (cells) in a complex biological organism. Each component is a simple agent or a complex multi-agent system. The agent model is

expressed as a computational paradigm using two specific mechanisms: membrane systems and communicating X-machine systems.

These approaches have some similarities: they are computational devices pointing out components running in a fully parallel manner and acting as a society of agents For both the design is flexible and reusable as the whole system may be built from individual components (mutisets and rules for membrane approach and component X-machines for communicating X-machine part) that are than 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 [13], the memory is implicitly defined as being the set of symbols associated to each component. For communicating X-machines the inputs and the outputs are explicitly associated to every basic function from Φ and a memory element is part of any computation. The former approach shows a synchronous parallel behaviour while the components of the later one work fully asynchronously. The communications between two components (a sender and a receiver) are initiated and lead by the message's sender, in the membrane case, and through a synchronized behaviour of some functions defined in both components, in the other case. For modelling the change of status for a foraging bee, in the membrane based approach, the same rules are used but with different memory symbols, whereas in the other approach the whole structure should be considered.

As it may be noticed the two approaches point out similarities but also important differences making them complementary ways of defining agent-based organisms. At some point we may also contemplate the idea of combining these approaches into a coherent more complex model. Such an attempt has been made for the sequential variant for the X-machines when the basic functions have been replaced by some rewriting rules of a distributed grammar system [4].

Further work is required to animate the agent-based specifications such as to compare the behaviour of these models with the data already obtained through some experiments [14]. In this respect tools are under construction for both approaches and results are expected to be reported soon.

REFERENCES

[1] 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.

[2] E. Csuhaj-Varju, J. Dassow, J. Kelemen, Gh. Paun, Grammar Systems. A Grammatical Approach to Distribution and Cooperation, Gordon and Breach, London, 1994.

[3] T.Finin, Y.Labrou, J.Mayfield, KQML as an Agent Communication Language. In: Software Agents, J.M. Bradshaw (Ed.), Menlo Park, Calif., AAAI Press, 291-316, 1997

[4] M Gheorghe, Generalised Stream X-machines and Cooperating Distributed Grammar Systems, Formal Aspects of Computing, 2001 (in press).

[5] M. Holcombe, Modelling cellular systems: the post-genome challenge, UCL London 2001, submitted to Briefings in Bioinformatics.

[6] M. Holcombe, F. Ipate, Correct Systems. Building a Business Process Solution, Springer, 1998.

[7] E.Kapeti, P.Kefalas, A Design Language and Tool for X-Machine Specification, In Advances in Informatics (D.I.Fotiadis, S.D.Nikolopoulos eds.), World Scientific Publishing Company, 134-145, 2000

[8] P. Kefalas, Using Communicating X-Machines to Model Social Insects Behaviour, Technical Report, Dept. of Computer Science, City College, Thessaloniki, 2001

[9] P. Kefalas, G, Eleftherakis, E. Kehris, Modular Modeling of Large-Scale Systems using Communicating X-Machines, In 8th Panhellenic Conference of Informatics 2001 (to appear).

[10] G. Paun, Computing with Membranes, J of Computer and System Sciences, 61, 1(2000), 108-143.

[11] Grammatical models of multi-agent systems, (Gh. Paun, A. Salomaa eds), Topics in Computer Mathematics 8, Gordon and Breach Science, Amsterdam, 1999.

[12] A. S. Rao and M. P. Georgeff, BDI Agents: From Theory to Practice, in Proceedings of the 1st International Conference on Multi-Agent Systems, San Francisco, USA, 1995, 312-319.

[13] Y Suzuki, H Tanaka, Chemical evolution among artificial proto-cells, Artificial Life VII, 54-63, 2000.

[14] H de Vries, J.C. Biesmeijer, Modelling collective foraging by means of individual behaviour rules in honey-bees, Behav Ecol Sociobiol, 44 (1998), 109-124.

[15] Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence, (G Weiss ed), The MIT Press.

[16] M. Wooldridge, N.R. Jennings, Intelligent Agents - Theory and Practice, Knowledge Engineering Review, 10, 2(1995), 115-152.