International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:1, 2007

Modeling Biology Inspired Reactive Agents Using X-machines

George Eleftherakis, Petros Kefalas, Anna Sotiriadou, and Evangelos Kehris

***Abstract*—**Recent advances in both the testing and verification of software based on formal specifications of the system to be built have reached a point where the ideas can be applied in a powerful way in the design of agent-based systems. The software engineering research has highlighted a number of important issues: the importance of the type of modeling technique used; the careful design of the model to enable powerful testing techniques to be used; the automated verification of the behavioural properties of the system; the need to provide a mechanism for translating the formal models into executable software in a simple and transparent way. This paper introduces the use of the X-machine formalism as a tool for modeling biology inspired agents proposing the use of the techniques built around X-machine models for the construction of effective, and reliable agent-based software systems.

Keywords— Biology Inspired Agent, Formal Methods, X-machine

I. Introduction

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 [1]. 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 [2]. The environment itself must be defined in some precise, mathematical way. 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, and
- 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 Holcombe and Ipate [2], 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, and
- 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*.

Although agent-oriented software engineering aims to manage the inherent complexity of software systems [3], there is still no evidence to suggest that any method proposed leads towards "correct" systems. In the last few decades, there has been a strong debate on whether *formal methods* can achieve this goal. Academics and practitioners adopted extreme positions either for or against formal methods [4]. It is, however, apparent that the truth lies somewhere between and that there is a need for use of formal methods in software engineering in general [5], while there are several specific cases proving the applicability of formal methods in agent development, as we shall see in the next section.

Software system specification has centred on the use of models of data types, either functional or relational models such as *Z* or *VDM* or axiomatic ones such as *OBJ*. Although these have led to some considerable advances in software design, they lack the ability to express the dynamics of the system. Also, transforming an implicit formal description into an effective working system is not straightforward. Other formal methods, such as *Finite State Machines* or *Petri Nets* capture the essential feature, which is "change", but fail to describe the system completely, since there is little or no reference at all to the internal data and how this data is affected by each operation in the state transition diagram. Other methods, like *Statecharts*, capture the requirements of dynamic behaviour and modelling of data but are rather informal with respect to clarity and semantics. So far, little attention has been paid in formal methods that could facilitate all crucial stages of "correct" system development, modelling, verification and testing. This paper will introduce such a

Manuscript received November 5, 2004.

George Eleftherakis is with the Computer Science Department, CITY college Thessaloniki, Greece, Affiliated Institution of the University of Sheffiled, UK (corresponding author; phone: +30-2310-275575; fax: +30-2310-287564; e-mail: eleftherakis@city.academic.gr).

Petros Kefalas is with the Computer Science Department, CITY college Thessaloniki, Greece, Affiliated Institution of the University of Sheffiled, UK (e-mail: kefalas@city.academic.gr).

Anna Sotiriadou is with the Computer Science Department, CITY college Thessaloniki, Greece, Affiliated Institution of the University of Sheffiled, UK (e-mail: sotiriadou@city.academic.gr).

Evangelos Kehris is with the Technological Education Institute (TEI) of Serres, Greece (e-mail: kehris@teiser.gr).

formal method, namely X-machines, which closely suits the needs of agent development, while at the same time being intuitive and practical.

II. Modeling Reactive Agents

A. X-machine

A X-machine is a general computational machine introduced by Eilenberg [6] and extended by Holcombe [7] that resembles a Finite State Machine (FSM) but with two significant differences:

- there is memory attached to the machine, and
- the transitions are not labeled with simple inputs but with functions that operate on inputs and memory values.

These differences allow the X-machines to be more expressive and flexible than the FSM. Other machine models like pushdown automata or Turing machines are too low level and hence of little use for specification of real systems. X-machines employ a diagrammatic approach of modelling the control by extending the expressive power of the FSM. They are capable of modelling both the data and the control of a system. Data is held in memory, which is attached to the X-machine. Transitions between states are performed through the application of functions, which are written in a formal notation and model the processing of the data. Functions receive input symbols and memory values, and produce output while modifying the memory values (Fig.1). 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 [2] is an 8-tuple $M = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$, 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 machine M, 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.

X-machines can be used as a core method for an integrated formal methodology of developing correct systems. The X-machine integrates both the control and data processing while allowing them to be described separately.

The X-machine formal method forms the basis for a specification/modelling language with a great potential value to software engineers. It is rather intuitive, while at the same time formal descriptions of data types and functions can be written in any known mathematical notation. Finally, X-machines can be extended by adding new features to the original model, such as hierarchical decomposition and communication, which will be described later. Such features are particularly interesting in agent-based systems.

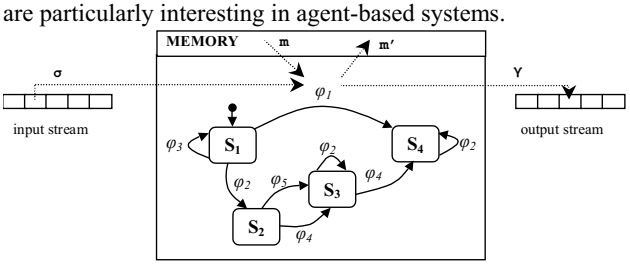

Fig. 1. An abstract example of a X-machine; φ_i: functions operating on inputs and memory, S_i: states. The general formal of functions is: $\varphi(\sigma,m) = (\gamma,m')$ *if condition*

B. Modeling Biology Inspired Reactive Agents

Many biological processes seem to behave like agents, as for example a colony of ants. Much research has been based on such behaviour in order to solve interesting problems [8]. An important task of some ants is to find food and carry it to its nest. This can be accomplished by searching for food at random or by following pheromone trails that other ants have left on their return back to the nest [9]. While moving, an ant should avoid obstacles. Once food is found, an ant should leave a pheromone trail while travelling back to its nest, thus implicitly communicating with other ants the destination of a source where food may be found. When the nest is found, the ant drops the food. Clearly, this is a reactive agent that receives inputs from the environment and acts upon these inputs according to the state in which the agent is. Such reactive agents can be fairly easily modelled by a FSM in a rather straightforward way by specifying the states and the inputs (percepts) to be used for state transitions (Fig.2).

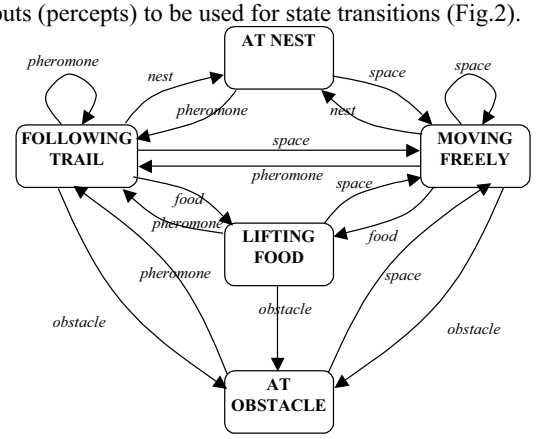

Fig. 2: A Finite State Machine modeling an ant's behaviour

The FSM lacks the ability to model any non-trivial data structures. In more complex tasks, one can imagine that the actions of the agents will also be determined by the values stored in its memory. For example, an agent may know its position, remember the position of the food source or the position of obstacles, thus building a map of the environment in order to make the task eventually more efficient. Using FSM or variants of it [10], [11] for such agents is rather complicated since the number of states increases in

combinatorial fashion to the possible values of the memory structure. X-machines can facilitate modelling of agents that demand remembering as well as reactiveness. Fig.3 shows the model of an ant that searches for food, but also remembers food positions in order to set up its next goals. The behaviour of obstacle avoidance is omitted for simplicity.

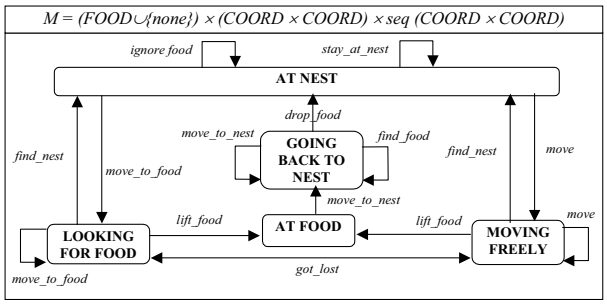

Fig. 3. A X-machine that models an ant

Formally, the definition of the X-machines requires all elements of the 8-tuple $(\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$. First of all, the input set consists of the percept and the x and y coordinate it is perceived:

$\Sigma = (\{space, nest\} \cup FOOD) \times COORD \times COORD$

where *[FOOD]* is a basic type and *COORD* is of type integer, $COORD \subseteq Z$.

The set of outputs is defined as a set of messages:

$\Gamma = \{$"moving freely", "moving to nest", "dropping food", ...$\}$

The states in which the agents can be are five:

$Q = \{$*At Nest, Moving Freely, At Food, Going Back To Nest, Looking For Food*$\}$.

The state "*Moving Freely*" applies to an agent that does not have a specific goal and searches in random for a food source. The state "*Going Back To Nest*" applies when the agent is carrying a food item and it is on its way back to its nest. The state "*Looking For Food*" applies when the agent has a goal, i.e. remembers where food is found during previous explorations of the terrain.

The memory consists of three elements, i.e. what the agent carries, the current position of the agent, and the sequence of positions where food is found during its exploration:

$M = (FOOD \cup \{none\}) \times (COORD \times COORD) \times seq\ (COORD \times COORD)$

where *none* indicates that no food is carried.

The initial memory and the initial states are respectively:

$m_0 = (none, (0,0), nil)$

$q_0 =$ "At Nest"

It is assumed that the nest is at position (0,0).

The next state partial function is depicted with the state diagram in Fig.3.

The type Φ is a set of functions of the form:

function_name(*input_tuple*, *memory_tuple*) → (*output*, *memory_tuple'*), if *condition*.

move((space,xs,ys), (none,(x,y),nil)) →
("moving freely",(none,(xs,ys),nil)),
if next(x,y,xs,ys)

move_to_food((space,xs,ys),(none,(x,y),<(fpx,fpy)::rest>)) →
("moving to food",(none,(nx,ny),<(fpx,fpy)::rest>)),
if next(x,y,xs,ys) ∧ loser_to_food(fpx,fpy,xs,ys)

move_to_nest((space,xs,ys), (food,(x,y),foodlist)) →
("moving to nest",(food,(nx,ny),foodlist)),
if food ∈ FOOD ∧ next(x,y,xs,ys) ∧ closer_to_nest(xs,ys)

lift_food((f,x,y),(none,(x,y),foodlist)) →
("lifting food",(f,(x,y),<(x,y) :: foodlist>)),
if f∈FOOD ∧ (x,y)∉foodlist

lift_food((f,x,y),(none,(x,y),foodlist)) →
("lifting food",(f,(x,y),foodlist)),
if f∈FOOD ∧ (x,y)∈foodlist

find_food((f,fpx,fpy), (food,(x,y),foodlist)) →
("more food",(food,(x,y),<(fpx,fpy)::foodlist>)),
if f∈FOOD ∧ f∉foodlist

drop_food((nest,0,0), (food,(x,y),foodlist)) →
("dropping food",(none,(0,0),foodlist))

find_nest((nest,0,0), (none,(x,y),foodlist)) →
("found nest again",(none,(0,0),foodlist))

got_lost((space,fpx,fpy), (none,(x,y),<(fpx,fpy)>)) →
("got lost",(none,(x,y),nil)),
if next(x,y,xs,ys)

ignore_food((food,0,0), (none,(0,0),foodlist)) →
("ignore food",(none,(0,0),foodlist)),
if f∈FOOD

stay_at_nest((nest,0,0), (none,(0,0),foodlist)) →
("staying in",(none,(0,0),foodlist))

where the functions *next*, *closer_to_nest* and *closer_to_food* are considered as external functions, i.e. functions that are defined elsewhere (possibly as X-machines themselves):

next: COORD× COORD× COORD× COORD→BOOLEAN
closer_to_nest: COORD× COORD →BOOLEAN
closer_to_food: COORD× COORD× COORD× COORD → BOOLEAN

III. Discussion

We have demonstrated how X-machines are able to model both the control and the data part of a complex system and therefore it possesses valuable characteristics that are desirable to software engineering of agent systems.

A framework for formal development of systems proposed in [12] uses X-machines as a formal modeling language. Using X-machines as the main core, the development of an agent can be mapped into several actions that are presented in the following paragraphs.

Developed X-machine models can be verified for desired properties an agent should possess, using a specifically designed formal verification technique to prove the validity of X-machine models [13]. Use of this formal verification technique (model checking) for X-machine models will increase the confidence that the proposed model has the desired characteristics. This technique enables the designer to automatically verify the developed model against temporal logic formulas that express the properties that the agent should have.

X-machines support not only static but also dynamic analysis. It is possible to use the formal testing strategy described in [2] to test the implementation and prove its correctness with respect to the X-machine model. Formally testing the implementation against the verified model it is possible to assure that all desired properties of an agent hold in the final product, increasing the confidence in using the final product.

The formalism is also enhanced with a methodology for building communicating systems out of existing components, i.e. stand-alone X-machines, which allows a disciplined development of large systems. The approach is practical, in the sense that the software engineer can separately specify the components and then describe the way in which these components communicate. Also, X-machine models can be re-used in other systems, since the only thing that needs to be changed is the communication part. The major advantage is that the methodology also lends itself to modular testing strategies in which X-machines are individually tested as components while communication is tested separately. It is found that by using communicating X-machines, we can formally model multi-agent systems, with agents modeled as an aggregate of different communicating behaviours [14].

A set of tools for X-machines exists and it is developed based on the X-machine Description Language (XMDL) to code X-machine models. With the use of tools that are built around the XMDL language it is possible to syntactically check the model and then automatically animate it [15]. Through this simulation it is possible for the developers to informally verify that the model corresponds to the actual system under development and also to demonstrate the model to the end-users aiding them to identify any misconceptions regarding the user requirements between them and the development team.

IV. Conclusion

The X-machine formal method is valuable to agent developers since it is rather intuitive, while at the same time can model non-trivial data structures, offering characteristics desirable to agent-oriented software engineering for developing "correct" systems, as discussed in this paper. With the continuous verification and testing from the early stages, risks are reduced and the developer is confident of the correctness of the system under development throughout the whole process. It is worth noticing that components that have been verified and tested can be reused without any other quality check in the proposed communicating X-machine system supported by the methodology is based in the idea of reusability, thus minimising the development time without risking the quality of the product.

Future work includes extension of the communicating system to support models with dynamic behaviour, i.e. self-reconfigurable multi-agent systems [16].

Acknowledgment

We would like to thank Prof Mike Holcombe for his valuable comments on our work over the last years.

References

[1] N.R. Jennings, "On agent-based software engineering", *Artificial Intelligence*, vol. 117, pp.277-296, 2000.
[2] M. Holcombe and F. Ipate, *Correct systems: Building a business process Solution*. Springer Verlag, London, 1998.
[3] M. Wooldridge and P. Ciancarini, "Agent-oriented software engineering: The state of the art", in *Proc. First Int. Workshop on Agent-Oriented Software Engineering*, pp.1-28, 2000.
[4] W. D. Young, "Formal Methods versus Software Engineering: Is There a Conflict?", In *Proceedings of the Fourth Testing, Analysis, and Verification Symposium*, pp. 188-899, 1991.
[5] E. Clarke and J. M. Wing, "Formal Methods: State of the Art and Future Directions", *ACM Computing Surveys*, vol. 28, no.4, pp.626-643, 1996.
[6] S. Eilenberg. *Automata, Machines and Languages*. Vol. A. Academic Press, 1974.
[7] M. Holcombe, "X-machines as a basis for dynamic system specification", *Software Engineering Journal*, vol. 3, no.2, pp. 69-76, 1988.
[8] M. Dorigo and G. Di Caro, "The ant colony optimization meta-heuristic", In D. Corne, M.Dorigo, & F. Glover (Eds.), New Ideas in Optimization, pp.11-32, McGraw-Hill, 1999.
[9] J. L. Deneubourg, S. Aron, S. Goss, and J.M. Pasteels, "The self-organizing exploratory pattern of the Argentine ant", *Journal of Insect Behavior*, vol. 3, pp. 159-68, 1990.
[10] R. A. Brooks, "A robust layered control system for a mobile robot", *IEEE Journal of Robotics Automation*., vol. 2, no. 7, pp.14-23, 1986.
[11] S.R. Rosenschein and L.P. Kaebling, "A situated view of representation and control", *Artificial Intelligence*, vol.73, no.1-2, pp 149-173, 1995.
[12] G. Eleftherakis and A.J. Cowling, "An Agile Formal Development Methodology", In *1st South Eastern European workshop on Formal Methods (SEEFM 03)*, pp. 36-47, Thessaloniki, November 2003.
[13] G. Eleftherakis, P. Kefalas, and A. Sotiriadou, "Formal Verification of Agent Models", In I.P.Vlahavas and C.D.Spyropoulos (eds), *Proceedings of the 2nd Hellenic Conference on AI (SETN02)*, pp.425-435, 2002.
[14] P. Kefalas, G. Eleftherakis, and E. Kehris, "Communicating X-machines: a practical approach for formal and modular specification of large systems", *Information and Software Technology*, vol. 45, no.5, pp.269-280, April 2003.
[15] P. Kefalas, G. Eleftherakis, and A. Sotiriadou, "Developing Tools for Formal Methods", In *9th Panhellenic Conference on Informatics*, pp. 625-639, Thessaloniki, November 2003.
[16] P. Kefalas, G. Eleftherakis, M. Holcombe, I. Stamatopoulou, "Formal Modelling of the Dynamic Behaviour of Biology-Inspired Agent-Based Systems", in *Molecular Computation Models: Unconventional Approaches*, M.Gheorghe (ed), Idea Publishing Group (IDG), 2005.