

Formal Modelling of Reactive Agents as an Aggregation of simple Behaviours

Petros Kefalas

Dept. of Computer Science, CITY Liberal Studies,
Affiliated Institution of the University of Sheffield
13 Tsimiski Street, Thessaloniki 546 24, Greece
kefalas@city.academic.gr

Abstract. Agents, as highly dynamic systems, are concerned with three essential factors: (i) a set of appropriate environmental stimuli, (ii) a set of internal states, and (iii) a set of rules that relates the previous two and determines what the agent state will change to if a particular stimulus arrives while the agent is in a particular state. Although agent-oriented software engineering aims to manage the inherent complexity of software systems, there is still no evidence to suggest that any proposed methodology leads towards correct systems. In the last few decades, there has been a strong debate on whether formal methods can achieve this goal. In this paper, we show how a formal method, namely X-machines, can deal successfully with agent modelling. The X-machine possesses all those characteristics that can lead towards the development of correct systems. X-machines are capable of modelling both the changes that appear in an agent's internal state as well as the structure of its internal data. In addition, communicating X-machines can model agents that are viewed as an aggregation of different behaviours. The approach is practical and disciplined in the sense that the designer can separately model the individual behaviours of an agent and then describe the way in which these communicate. The effectiveness of the approach is demonstrated through an example of a situated, behaviour-based agent.

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 [1]. Agents, as highly dynamic systems, are concerned with three essential factors: (i) a set of appropriate environmental stimuli or inputs, (ii) a set of internal states of the agent, and (iii) a set of rules that relate the two above and determines what the agent state will change to if a particular stimulus arrives while the agent is in a particular state.

Although *agent-oriented software engineering* aims to manage the inherent complexity of software systems [2], there is still no evidence to suggest that any methodology proposed so far 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 [3]. It is, however, apparent that the truth lies somewhere in between and that there is a need for use of formal methods in software engineering in general [4], while there are several specific cases proving the applicability of formal methods in agent development. One of them was to formalize *PRS* (Procedural Reasoning System), a variant of the *BDI* architecture [5] through 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 [6]. In an attempt to capture the dynamics of an agent system, an agent can be viewed as a situated automaton that generates a mapping from inputs to outputs, mediated by its internal state [7]. Alternatively, the *DESIRE* framework focuses on the specification of the dynamics of the reasoning and acting behaviour of multi-agent systems [8]. Finally, in a less formal approach, extensions to *UML* were proposed (*AUML*) in order to accommodate the distinctive requirements of agents [9].

Although all the above have contributed to formal modelling of intelligent agents, they were not able to solve the complexity problem of a single or a multi-agent system. In this paper, we describe a formal method for modelling agents through its behaviours. We decompose modelling into simpler independent steps that facilitates and simplifies the development process. In section 2, the motivation of our work is given and the background theory is introduced. Section 3 defines the proposed formal method and section 4 demonstrates its capability to express complex systems through an example. The practical advantages and the evaluation of the method are discussed in Section 5. Finally, section 5 concludes this paper by presenting further work.

2 Motivation

One of the challenges that emerge in *intelligent agent engineering* is to develop agent models and agent implementations that are correct. The criteria for correctness, as stated in [10], are: (i) the initial agent model should match with the requirements, (ii) the agent model should satisfy any necessary properties in order to meet its design objectives, and (iii) 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*. Proving correctness is facilitated when modelling of an agent is done in a formal way. So far, however, little attention has been paid in formal methods that could aid all crucial stages of correct system development. The main reason for this drawback of formal methods is that they focus on one part of the system modelling only. For example, system specification has centered on the use of models of data types, either functional or relational models such as *Z* [11] or *VDM* [12]. Although these have led to some considerable advances in software design, they lack the ability to express the dynamics of the system. Other formal methods, such as *Finite State Machines* [13] or *Petri Nets* [14] have little or no reference at all to the internal data and how this data is affected by each operation in the state transition diagram. Finally, *Statecharts* [15] capture the requirements of dynamic behaviour and modelling of data but are rather informal with respect to clarity and semantics.

performance. Similarly, in the *Cassiopeia* method [19], the agents are defined by following three steps: (i) identifying the elementary behaviours that are implied by the overall task, (ii) identifying the relationship between elementary behaviours, and (iii) identifying the organizational behaviours of the system.

Based on the latter, we believe that such a methodology can be practical in developing agent models only if it can be able to cope with modelling of the behaviours separately from the description of the behaviour interaction. This approach has several advantages for the developer who: (i) does not need to model a new agent from scratch, (ii) can re-use existing behaviours in other agent models, and (iii) can view agent modelling as two separate distinct activities.

With respect to the above, certain approaches for building an agent require a brand new conceptualisation and development of the system as a whole. This approach has a major drawback, i.e. one cannot re-use existing behaviour models that have been already verified and tested for their correctness. Often, in agent systems, components from other agents are required. A desirable approach would be to conceptualise an agent as an aggregation of independent smaller models of behaviours, which need to communicate with each other. Thus, one does not need to worry about the individual components, in which *verification* and *testing* techniques are applied, but only with appropriately linking those components. This would lead to a disciplined development methodology, which implies two distinct and largely independent development activities, i.e. building simple models and then employing communication between them.

3 A Formal Method for Agent Modelling

Bearing in mind the above, a formal method in order to be useful to modelling of intelligent agents should be able:

- to model both the data and the changes of an agent,
- to model separately the behaviours of an agent and the ways the behaviours interact with each other,
- to be intuitive, practical and effective towards implementation of an agent, and
- to facilitate development of correct agents.

All the above are prominent characteristics of the X-machine. A X-machine is a general computational machine [20,21] that resembles a FSM but with two significant differences: (i) there is memory attached to the machine, and (ii) the transitions are not labeled with simple inputs but with functions that operate on inputs and memory values. X-machines employ a diagrammatic approach of modelling the control by extending the expressive power of the FSM. 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.2). 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 [22] is an 8-tuple $XM = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$, 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 XM , 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.

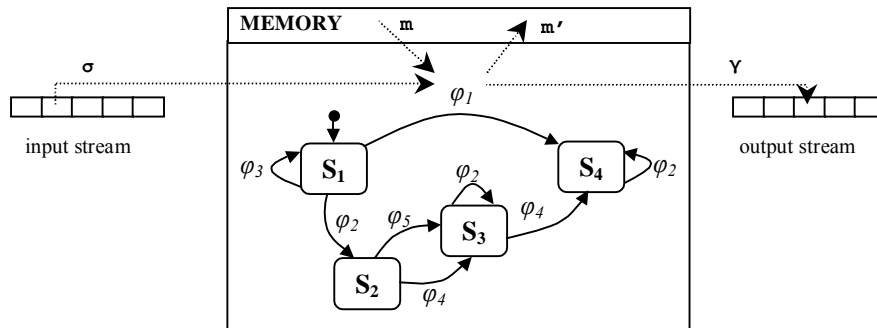


Fig. 2. An abstract example of a X-machine; φ_i functions operating on inputs and memory, S_i states. The general formal of functions is: $\varphi(\sigma, m) = (\gamma, m')$ if condition

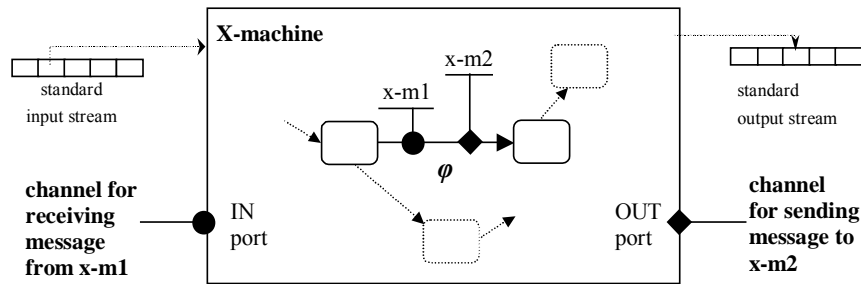


Fig. 3. An abstract example of a Communicating X-machine component.

Several theoretical approaches for communicating X-machines have been proposed [23,24,25,26]. All of them lead to the development of large-scale systems as a set of X-Machines that communicate with each other. In this section we will describe the approach that focuses on the practical development of communicating systems but also subsumes all others [26].

The functions of a X-machine, if so annotated, 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.3 accepts its input from the model *x-m1* and writes its output to model *x-m2*. Multiple communications channels for a single X-machine component may exist.

4 Modelling Reactive Agents as X-machines

Consider a reactive agent that is working in some environment and collects objects, which are then carried to its base. Initially, the agent knows no object and searches in random to find some of them. When an object is found, the agent carries it to its base, but while going back records the objects met on the way in its memory. When the object is dropped at the base, the agent sets an object from its list of objects as the current goal and looks for it, thus performing a directed search. At all times the agent is able to avoid obstacles found on its way. The agent is modelled by the X-machine shown in Fig.4., which shows the next state partial function.

First of all, the input set of the X-machine consists of the percept and the *x* and *y* coordinate it is perceived:

$$\Sigma = (\{space, base\} \cup OBSTACLE \cup OBJECT) \times COORD \times COORD,$$

where $[OBSTACLE, OBJECT]$ are basic types and *COORD* is of type *integer*, that is $COORD \subseteq \mathbb{Z}$. A basic type is the kind of abstraction made in a specification language in the attempt to define "anything", without getting into details concerning their implementation. The set of outputs is defined as a set of messages:

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

The states in which the agents can be are:

$$Q = \{ At\ Base, Searching, At\ Obstacle, Going\ Back, Directed\ to\ Object \}.$$

The state "*Searching*" applies to an agent that does not have a specific goal and searches in random for an object. The state "*Going Back*" applies when the agent is carrying an object and it is on its way back to its nest. The state "*Directed to Object*" applies when the agent has a goal, i.e. remembers where an object 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 objects are found during its exploration:

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

where *none* indicates that no object is carried. The initial memory and the initial states are respectively $m_0 = (none, (0,0), nil)$ and $q_0 = "At\ Base"$. It is assumed that the base is at position $(0,0)$. The type Φ is a set of functions of the form:

$$function_name(input, memory) \rightarrow (output, memory'), \text{ if condition.}$$

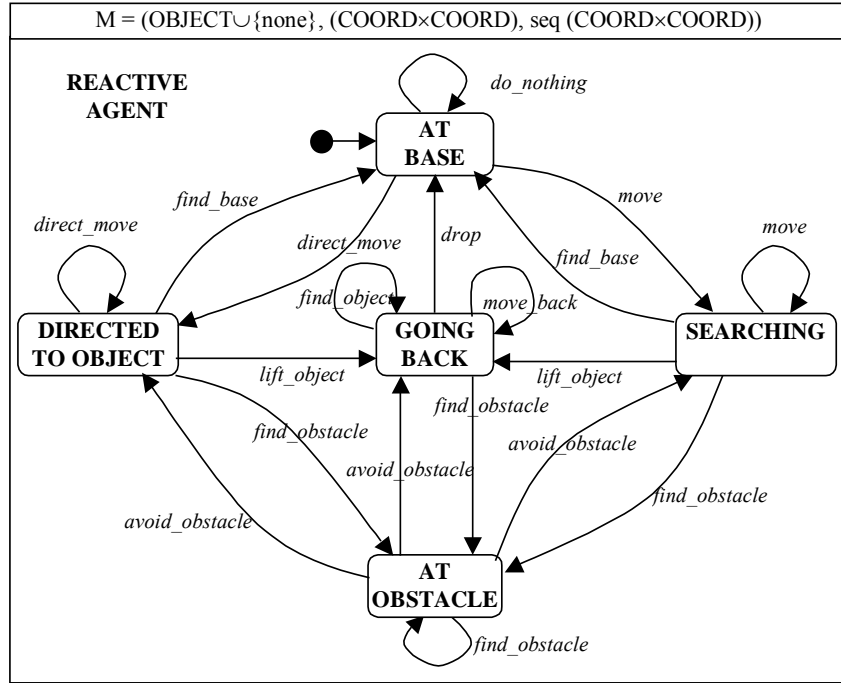


Fig. 4. An example of a reactive agent modelled as X-machine.

For example, the following are some of the functions of the agent model:

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

$direct_move((space, xs, ys), (none, (x, y), <px, py>::rest)) \rightarrow$
 $("moving\ to\ object", (none, (nx, ny), <px, py>::rest)),$
 $if\ next(x, y, xs, ys) \wedge closer_to_object(px, py, xs, ys)$

$lift_object((obj, x, y), (none, (x, y), objectlist)) \rightarrow$
 $("lifting\ object", (obj, (x, y), <(x, y) :: objectlist>)),$
 $if\ obj \in OBJECT$

$find_object((obj, x, y), (item, (x, y), objectlist)) \rightarrow$
 $("record\ object\ position", (item, (x, y), <(x, y) :: objectlist>)),$
 $if\ item \neq none \wedge obj \in OBJECT \wedge (x, y) \notin objectlist$

where the functions $next$, $closer_to_object$ are considered as external functions:

$next: COORD \times COORD \times COORD \times COORD \rightarrow BOOLEAN$

$closer_to_object: COORD \times COORD \rightarrow BOOLEAN$

External functions are functions that are already defined elsewhere or they are possibly X-machines themselves. The X-machine theory allows for hierarchical refinement of models, and therefore a X-machine can be considered as a function that can be used in modelling of other X-machines [10].

An agent, however, can be conceptualized as a set of simple behaviours. The methodology for building such an agent model should be based around three steps: (a)

identification of the behaviours, (b) modeling of individual behaviours, and (c) aggregation of behaviours in order to construct the whole agent model. Towards this approach, communicating X-machines can be used.

The behaviours of the previously described are: (a) searching for an object, (b) moving directly to an object, (c) lifting and dropping objects, (d) avoiding obstacles, (e) traveling back to the base, (f) building a map of the environment.

Each behaviour can be modelled as simple X-machines, as long as the states, the input set (percepts in which this behaviour will react) and the memory is determined. For example, “moving directly to an object” is an X-machine (Fig.5), called *MD*, with three states $Q=\{\text{“Waiting for Goal”}, \text{“Starting Search”}, \text{“Moving Directly”}\}$, an input set $\Sigma=\{\text{space}, \text{base}\}$, a memory $M=(\text{agent_position}, \text{object_position})$, an initial state $q_0=\text{“Waiting for Goal”}$, etc. Similarly, “building map of the environment” is an X-machine (Fig.5), called *BEM*, with only one state $Q=\{\text{“Building Map”}\}$, an input set $\Sigma=\text{OBJECT}\times\text{COORD}\times\text{COORD}$, a memory $M=(\text{seq COORD}\times\text{COORD})$, which holds the objects positions, etc. The rest of the behaviours can be modelled accordingly. Each machine has different memory, inputs (percepts), and functions. Symbols used for inputs a memory need not be the same, i.e. models can be independently built without any reference to other models at this stage.

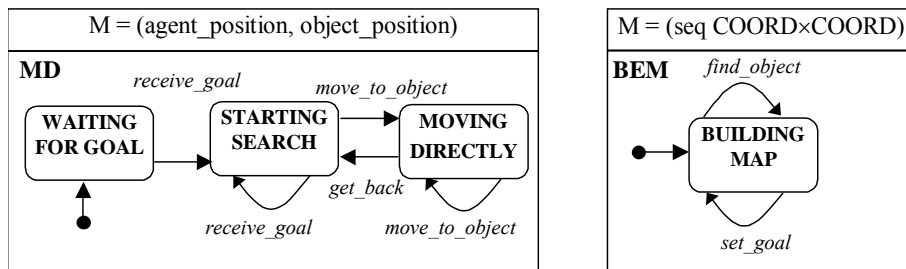


Fig. 5. The behaviours of an agent “moving directly to an object” and “building map of the environment” modelled separately as X-machine components.

The next task is to set up the communication between the components. For example, the behaviour of building the map of the environment should set a goal to the behaviour of moving directly to an object. This is done by utilising the notation of communicating X-machines, as shown in Fig.6. When an object is found, *find_object* is applied and the memory of *BEM* is updated. The *set_goal* function sends a message of type $\text{COORD}\times\text{COORD}$ to *MD*, which is perceived as input from *receive_goal*. The memory (*object_position*) is updated. Model *MD* continues to receive inputs and apply its functions. In the meantime, if other objects are found, their position is communicated from *BEM* to *MD* through the communication channel between the two X-machines.

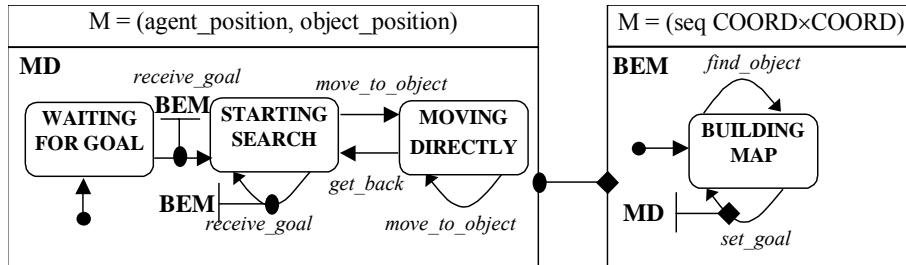


Fig. 6. Communicating agent behaviours modelled as X-machines.

In the same manner, the rest of the behaviours can participate in the complete agent model, which is built of simple but communicating X-machines components (Fig.7). Each machine “works” separately and concurrently in an asynchronous manner. Each machine may read inputs from a communication channel instead of its standard input tape. Also, each machine may send a message through a communication channel that acts as input to functions of another component.

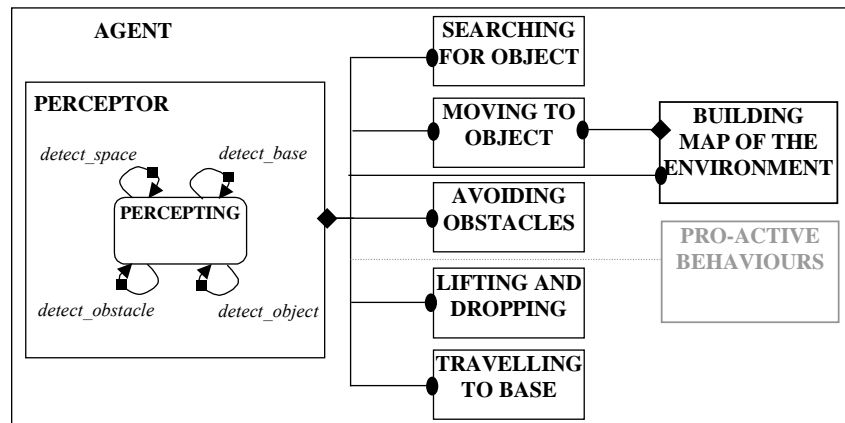


Fig. 7. The complete agent model modelled as an aggregation of different behaviours.

Modelling of an agent can be incremental by providing components, which will advance further the level of intelligent behaviour. More complex behaviours can be modelled, as for example, the X-machine that builds an environment map for free positions and positions of obstacles. Information held in the memory of this machine could be used to efficiently move around the environment, or even to model a pro-active behaviour for the agent (Fig.7). In principle, one can use X-machines to model even more complex behaviours, like planning. Although the modelling complexity is high, the theory behind X-machines allows modelling through hierarchical development, refinement and use of communicating X-machines for building large-scale systems [10].

5 Discussion and Evaluation

The approach described in this paper aims towards the development of formal models of situated agents and meets the requirements set up in section 2. 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 [10]. Firstly, it is the environment itself, which could be precisely or ill-specified or even completely unknown, but nevertheless it involves identifying the important aspects of the environment and the way in which they may change in accordance with the activities of the agent. And also, it is the agent, which 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. These concepts are captured by X-machines as demonstrated by the example presented. The X-machine method is rather intuitive, while formal descriptions of data types and functions can be expressed in any known mathematical notation.

An important issue in the use of X-machines as a modelling formal method is that they can lead towards the development of correct agents. Having constructed a model of an agent as a X-machine, it is possible to apply existing *model checking* techniques to verify its properties. A specifically defined logic, namely *XmCTL*, can verify the model expressed as X-machine against the requirements, since it can prove that certain properties, which implicitly reside on the memory of X-machine are true [27]. In addition, having ensured that the model is valid, we need to also ensure that the implementation is correct, this time with respect to the model. This can be achieved through a *complete testing strategy*, such as the one presented in [10], which finds all faults in the implementation. Therefore, X-machines can be used as a core method for an integrated formal methodology of developing correct systems.

By viewing an agent as an aggregation of behaviours, communicating X-machines can be used. This approach is disciplined, in the sense that the developer can separately model the components of an agent and then describe the way in which these components-behaviours communicate. Also, components can be re-used in other systems, since the only thing that needs to be changed is the communication part. For example, the behaviour for avoiding obstacles is a component of any robotic agent. The major advantage is that the methodology also lends itself to modular model checking and testing strategies in which X-machines are individually tested as components while communication is tested separately with existing methodologies, mentioned above.

Communicating X-machines may be used to model multi-agent systems. Modelling of multi-agent systems imposes the consideration of the means of communication between agents, in order to coordinate tasks, cooperate etc. Also, modelling of artificial environments in which agents act imposes the need of interaction between agents and the environment. These requirements are met by the communicating X-machines and shown to be effective in modelling of biology-inspired agents, such as a colony of ants or bees for collective foraging [28,29].

Finally, through the use of *XMDL* [30], the X-machine formal method aims to overcome one of the main criticisms for formal methods, i.e. practicality. *XMDL* (X-Machine Description Language) is a declarative mark-up language, which permits the

designer to write ASCII code in order to describe a X-machine model, rather than using any other ad-hoc mathematical notation. A number of tools have been developed, like test-case generators, model checkers, automatic translators to executable languages, such as *Prolog* or *Java*, animators etc., with *XMDL* as a common modelling language.

6 Conclusions and Further Work

We have presented a formal way to model behaviour-based agents, through the use of X-machines. X-machines provide the ability to model individual behaviours of an agent and then describe the way in which these behaviours can communicate with each other. The main advantage of using the particular formal method is that it can model both the internal changes of an agent state as well as the information stored in an agent. Such features are particularly interesting in agent systems, since they act on a dynamic environment and perceive the changes of the environment as inputs that alter their internal states as well as their knowledge or beliefs. In addition, the formal method can lead towards the disciplined development of correct agents using formal verification and complete testing strategies.

Current and future work involves the applicability of the approach to simple robotic agents, through the automatic translation of X-machine models to a higher level programming language. Having based their design on X-machines we are trying to guarantee that on one hand they possess the desired properties through model checking, and on the other hand that they behave correctly under all circumstances through complete testing.

References

1. Jennings, N.R.: On agent-based software engineering. *Artificial Intelligence* 117 (2000) 277-296
2. Wooldridge, M., & Ciancarini, P.: Agent-oriented software engineering: The state of the art. To appear in the *Handbook of Software Engineering and Knowledge Engineering*, World Scientific Publishing Co. (2001)
3. Young, W. D.: Formal Methods versus Software Engineering: Is There a Conflict? In *Proceedings of the Fourth Testing, Analysis, and Verification Symposium* (1991) 188-899
4. Clarke, E., Wing, J. M.: Formal Methods: State of the Art and Future Directions, *ACM Computing Surveys* 28 (4) (1996) 26-643
5. Rao, A.S., & Georgeff, M.: BDI Agents: from theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)* (1995) 312-319
6. Inverno, d' M., Kinny, D., Luck, M., & Wooldridge, M.: A formal specification of dMARS. In: M.P.Singh, A.Rao, and M.J.Wooldridge (Eds.): *Intelligent Agents IV – LNAI*, Vol.1365. Springer-Verlag (1998) 155-176
7. Rosenschein, S. R., & Kaebling, L. P.: A situated view of representation and control. *Artificial Intelligence* 73 (1-2) (1995) 149-173
8. Brazier, F., Dunin-Keplicz, B., Jennings, N., & Treur, J.: Formal specification of multi-agent systems: a real-world case. *Proceedings of International Conference on Multi-Agent Systems (ICMAS'95)*, MIT Press (1995) 25-32

9. Odell, J., Parunak, H. V. D., & Bauer, B.: Extending UML for agents. In Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence (2000)
10. Holcombe, M., & Ipate, F.: Correct systems: Building a business process Solution. Springer Verlag (1998)
11. Spivey, M.: The Z notation: A reference manual. Prentice-Hall (1989)
12. Jones, C. B.: Systematic software development using VDM. 2nd edn. Prentice-Hall (1990)
13. Wulf, W. A., Shaw, M., Hilfinger, P.N., & Flon, L.: Fundamental structures of computer science. Addison-Wesley (1981)
14. Reisig, W.: Petri nets - an introduction. EATCS Monographs on Theoretical Computer Science 4. Springer-Verlag (1985)
15. Harel, D.: Statecharts: A visual approach to complex systems. Science of Computer Programming 8 (3) (1987)
16. Brooks, R. A.: A robust layered control system for a mobile robot. IEEE Journal of Robotics Automation 2 (7) (1986) 14-23
17. Jennings, N.R.: An agent-based approach for building complex software systems. Communications of the ACM 44 (4) (2001) 35-41
18. Brooks, R. A.: Intelligence without reason. In: J. Mylopoulos, & R. Reiter (Eds.): Proceedings of the 12th International Joint Conference on Artificial Intelligence, Morgan Kaufmann (1991) 569-595
19. Collinot, A., Drogul, A. & Benhamou, P.: Agent oriented design of a soccer robot team. In proceedings of the 2nd International Conference on Multi-Agent Systems (1996) 41-47
20. Eilenberg, S.: Automata, Machines and Languages. Vol. A. Academic Press (1974)
21. Holcombe, M.: X-machines as a basis for dynamic system specification. Software Engineering Journal 3 (2) (1988) 69-76
22. Ipate, F., & Holcombe, M.: Specification and testing using generalised machines: a presentation and a case study. Software Testing, Verification and Reliability 8 (1998) 61-81
23. Balanescu, T., Cowling, A.J., Gheorgescu, H., Gheorghe, M., Holcombe, M., & Vertan, C.: Communicating stream X-machines systems are no more than X-machines. Journal of Universal Computer Science 5 (9) (1999) 494-507
24. Cowling, A.J., Gheorgescu, H., & Vertan, C.: A structured way to use channels for communication in X-machines systems. Formal Aspects of Computing 12 (2000) 485-500
25. Barnard, J.: COMX: a design methodology using communicating X-machines. Journal of Information and Software Technology 40 (1998) 271-280
26. Kefalas, P., Eleftherakis, G., & Kehris, E.: Modular modelling of large-scale systems using communicating X-machines. In: Y. Manolopoulos & S. Evripidou (Eds.): Proceedings of the 8th Panhellenic Conference in Informatics, Greek Computer Society (2001) 20-29
27. Eleftherakis, G., & Kefalas P.: Towards model checking of finite state machines extended with memory through refinement. In: G. Antoniou, N. Mastorakis, & O. Panfilov (Eds.): Advances in Signal Processing and Computer Technologies, World Scientific and Engineering Society Press (2001) 321-326
28. Georghe, M., Holcombe, M., & Kefalas, P.: Computational models for collective foraging. BioSystems 61 (2001) 133-141
29. Kefalas P., Holcombe M., Eleftherakis G., Gheorghe M.: A formal method for the development of agent-based systems. In: V. Plekhanova & S. Wermter (Eds.): Intelligent Agent Software Engineering, Idea Group Publishing Hershey, PA, USA (2002) (to be published)
30. Kapeti, E., & Kefalas, P.: A design language and tool for X-machines specification. In: D.I. Fotiadis, S.D. Nikolopoulos (Eds.): Advances in Informatics. World Scientific Publishing Company (2000) 134-145