Formal Modelling of the Dynamic Behaviour of Biology-Inspired Agent-based Systems

P. Kefalas[1], G. Eleftherakis[1], M. Holcombe[2], I. Stamatopoulou[3]

1 Dept. of Computer Science, CITY College,
13 Tsimiski Street, Thessaloniki 546 24, Greece
{kefalas, eleftherakis}@city.academic.gr

2 Dept. of Computer Science, University of Sheffield,
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK
m.holcombe@dcs.shef.ac.uk

3 South-East European Research Centre,
17 Mitropoleos Street, Thessaloniki 546 24, Greece
istamatopoulou@seerc.info

Formal Modelling of the Dynamic Behaviour of Biology-Inspired Agent-based Systems

ABSTRACT

Multi-agent systems are highly dynamic since the agents' abilities and the system configuration often changes over time. In some ways, such multi-agent systems seem to behave like biological processes; new agents appear in the system, some other cease to exist, communication between agents changes. One of the challenges is to attempt to formally model the dynamic configuration of multi-agent systems. Towards this aim, we present a formal method, namely X-machines, which can be used to formally specify, verify and test individual agents. In addition, communicating X-machines provide a mechanism for allowing agents to communicate messages to each other. We utilize concepts from biological processes in order to identify and define a set of operations that are able to reconfigure a multi-agent system. We present an example, in which a biology-inspired system is incrementally built in order to meet our objective.

INTRODUCTION

An *agent* is an encapsulated computer system that is situated in some environment and is capable of flexible, autonomous action in that environment in order to meet its design objectives (Jennings, 2000). The extreme complexity of agent systems is due to substantial differences in attributes between their components, high computational power required for the processes within these components, huge volume of data manipulated by these processes and finally possibly extensive amount of communication in order to achieve coordination and collaboration. The use of a computational framework that is capable of modelling both the dynamic aspect (i.e. the continuous change of agents' states together with their communication) and the static aspect (i.e. the amount of knowledge and information available), will facilitate modelling and simulation of such complex systems.

The multi-agent paradigm can be further extended to include biology-inspired systems. Many biological processes seem to behave like multi-agent systems, as for example a colony of ants or bees, a flock of birds, tissue cells etc (Dorigo et al., 1996). The vast majority of

computational biological models are based on an assumed, fixed system structure that is not realistic. The concept of growth, division and differentiation of individual components (agents) and the communication between them should be addressed in order to create a complete biological system which is based on rules that are linked to the underlying biological mechanisms allowing the dynamic evolution.

For example, consider the case of a tissue, which consists of a number of cells. Each cell has its own evolution rules that allow it to grow, reproduce and die over time or under other specific circumstances. The cells are arranged in some two- or three-dimensional space, and this layout implies the way cells interact with others in the local neighbourhood. The rules of communication and exchange of messages are dependent on the particular system. Being a dynamic system, the structure of the tissue, that is the configuration of cells and therefore their interaction, may change over time, thus imposing a change in the global state of the system. New cells are born, others die, while some are attached to or detached from existing ones (Figure 1).

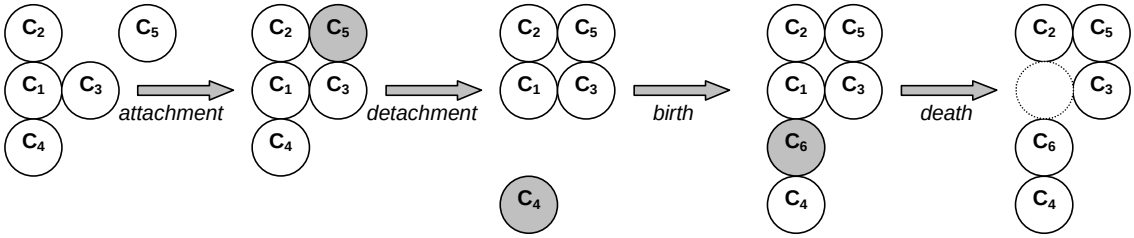

Figure 1: The dynamic evolution of a tissue consisting of individual cells

In the last years attempts have been made to devise computational models in the form of generative devices (Paun, 1998, 2002), (Banatre & LeMetayer, 1990), (Adleman, 1994), (Holcombe, 2001). In this chapter we focus on how we can model such dynamic multi-agent systems, taking as a principle example a biology-inspired system like the one presented above, by using a formal method, namely X-machines. A particular type of X-machines, called communicating X-machines (Kefalas et al., 2003b), will be used. X-machines and communicating X-machines have been successfully used in the past for the modelling of stand-alone as well as complex communicating systems (Kefalas et al., 2003a). We consider this formal method, not only as a theoretical tool for specification, but also as a practical tool

for modelling that eventually leads to the implementation of correct systems. The important aspect of using X-machines as a modelling method is that they support formal verification (Eleftherakis & Kefalas, 2001) and complete testing (Holcombe & Ipate, 1998). Thus, formal models expressed as X-machines can be checked as to whether they satisfy given properties through model checking, and the potential implementation can be tested against the model in order to identify errors. We will briefly present these issues in the discussion part of the chapter.

At least three benefits of using communicating X-machines for modelling multi-agent systems are emphasised:

- it is a natural environment in which to simulate dynamic systems,
- it is flexible enough so that when a new component is introduced into the play the others need not be restructured (this feature will become useful when simulating cell division, death etc.), and
- it is much more efficient than single X-machines initially used to simulate such systems.

This chapter will mainly concentrate on X-machines since it has been demonstrated that some variants of X-machines may become suitable for molecular computing models due to:

- their completeness property,
- capability of naturally simulating different models,
- suitability to define dynamic systems reacting to input stimuli,
- flexibility in expressing hierarchical or distributed systems, and
- ability in capturing hybrid specifications.

Our objectives are:

- to demonstrate that X-machines are suitable for modelling individual agents, by representing their internal data and knowledge, and how the stimuli received from the environment could change their internal state,
- to present the way in which communicating X-machines can facilitate modelling of multi-agent systems, but also their inadequacy to deal with the evolution of such systems,
- to identify and define a number of meta-operations that will enable the dynamic change of the configuration of multi-agent systems, and finally
- to suggest how dynamic multi-agent systems can be modelled as a whole.

One may argue that modelling multi-agent systems may require a new variant of X-machines that maps the particular requirements for such systems. However, the rationale behind using X-machines with their original definition is to be able to exploit to the maximum possible extent the legacy of X-machine theory concerning complete testing and formal verification. The particular requirements of multi-agent systems are dealt at a meta-level, thus not affecting modelling, testing and verification of the individual agent models.

The organization of this chapter is as follows: The first section presents a background on formal methods and their use in agent-based as well as biology-inspired systems. An analytical description of X-machines is given in the second section accompanied by the appropriate definitions and examples. In section three a number of definitions for operators, namely *COM*, *GEN*, *ATT*, *DET* and *DET* that could handle the dynamic structure of a multi-gent system are given. Also, two approaches for controlling the dynamic behaviour are discussed. Finally, the chapter concludes with a discussion on the benefits of using X-machines as a modelling tool.

BACKGROUND

A variety of formal methods have been used for software system specification. Specification has centred on the use of models of data types, either functional or relational models such as *Z* (Spivey, 1989) or *VDM* (Jones, 1990) or axiomatic ones such as *OBJ* (Futatsugi et al., 1985). 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* (Wulf et al., 1981) or *Petri Nets* (Reisig, 1985) 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* (Harel, 1987), capture the requirements of dynamic behaviour and modelling of data but are rather informal with respect to clarity and semantics.

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 formalise PRS (Procedural Reasoning System), a variant of the BDI architecture (Rao & Georgeff, 1995) with the use of Z, in order to understand an agent's architecture in a better way, to be able to move to the implementation through refinement of the specification and to develop proof theories for the architecture (D'Iverno et al. 1998). Trying to capture the dynamics of an agent system, Rosenschein & Kaebling (1995) viewed an agent as a situated automaton that generates a mapping from inputs to outputs, mediated by its internal state. Brazier et al. (1995) 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 (Benerecetti et al. 1999), (Rao & Georgeff, 1993). Towards implementation of agent systems, Attoui & Hasbani (1997) focused on program generation of reactive systems through a formal transformation process. A wider approach is taken by Fisher & Wooldridge (1997) who utilise Concurrent METATEM in order to formally specify multi-agent systems and then directly execute the specification while verifying important temporal properties of the system. Finally, in a less formal approach, extensions to UML to accommodate the distinctive requirements of agents (AUML) were proposed (Odell et al, 2000).

In the last years, some of the above formal approaches have been considered for modelling different biological phenomena and aspects of the living organisms. Others have attempted to devise new computational models, such as P Systems (Paun, 2000), originating their inspiration from bio-chemical processes that occur in living cells (the P Systems web page, 2003), (a bibliography of Molecular Computation and Splicing Systems Web Page, 2003) or Gamma (Banatre & Le Metayer, 1990) and Cham (Berry & Boudol, 1992) by modelling the concurrent behaviour of systems. There are also computational models like Boolean networks (Kauffman, 1993), developed to express bio-chemical reactions occurring at the cell level, or X-machines, utilized to model metabolic pathways (Holcombe, 1988), and the behaviour of bee and ant colonies (Gheorghe et al., 2001; Kefalas et al., 2003a). Hybrid approaches are developed to facilitate the specification of complex aspects related to micro or macro bio-structures (Duan et al., 1995; Balanescu et al., 2002; Gheorghe et al., 2003). Finally, attempts to show the equivalence of new computational models are made (Kefalas et al., 2003c).

MODELLING WITH COMMUNICATING X-MACHINES

X-machines

An *X-machine* is a general computational machine introduced by Eilenberg (1974) and extended by Holcombe (1988) that resembles a Finite State Machine (FSM) but with two significant differences:

- a memory is 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 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 for 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 the memory structure. 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 (Figure 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.

Definition. A *deterministic stream X-machine* (Holcombe & Ipate, 1998) is an 8-tuple $X = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$, where:

- Σ, Γ are the input and output finite alphabets respectively;
- Q is the finite set of states;
- M is the (possibly) infinite set called memory;
- Φ represents the type of the machine X, 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, $F: Q \times \Phi \rightarrow Q$, that given a state and a function from the type Φ, denotes the next state. F is often described as a state transition diagram;
- q_0 and m_0 are the initial state and initial memory, respectively.

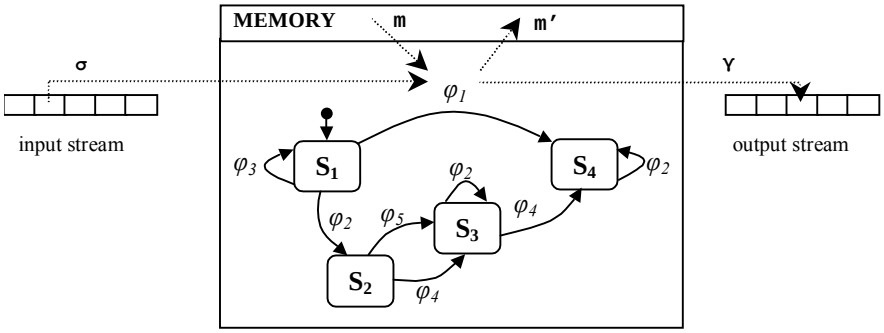

Figure 2**:** An abstract example of a X-machine; φ_i: functions operating on inputs and memory, S_i: states. The general format of functions is: $\varphi(\sigma, m) = (\gamma, m')$

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.

Modelling a Cell Agent as an X-machine

In this section, we are going to deal with the modelling of an agent as a stream X-machine. The model we present here is simple for reasons of exposition and it follows various underlying assumptions that entail an overall simplicity (e.g. discretisation of possible movements and directions, identical cells, two-dimensional plane etc). Consider a simple agent, e.g. a cell that is born, matures and eventually dies. Each of these life states can be described in terms of a state of an X-machine. While the agent is mature it is able to reproduce. The agent can move in four possible directions if it is forced to do so (i.e. by receiving an appropriate input). Naturally, the agent may exhibit other behaviours as well but for the sake of simplicity we will omit those. The interested reader may refer to (Kefalas, 2002) for further details. A state transition occurs when the appropriate input is received. The next state transition diagram, represented by *F*, is depicted in Figure 3. The definition of the cell formal model is:

$Q = \{born, matured, dead\}$;

$\Sigma = \{mature\_now, reproduce\_now, die\_now, north\_force, south\_force, west\_force, east\_force, accident\}$;

$\Gamma = \{$*"agent matures", "agent reproduces", "agent dies naturally", "mature agent dies by accident", "new born agent dies by accident", " north_force", "south_force", "west_force", "east_force"* $\}$;

$M = (X, Y), X, Y \in N_0$, where N_0 is the set of natural numbers including zero.

The initial memory represents the Cartesian coordinates, which describe the position of a cell in a two-dimensional plane (e.g. $m_0 = (16, 29)$).

$q_0 = born$;

$\Phi = \{$ *grows(mature_now , (X, Y)) = ("agent matures", (X, Y)),*
reproduces(reproduce_now, (X, Y)) = ("agent reproduces", (X, Y)),
dies(die_now, (X, Y)) = ("agent dies naturally", (X, Y)),
dies(accident, (X, Y)) = (" mature agent dies by accident", (X, Y)),
dies_suddenly(accident, (X, Y)) = ("new born agent dies by accident", (X, Y)),
move_east(west_force , (X, Y)) = ("west_force", (X+1, Y)),
move_west(east_force , (X, Y)) = ("east_force", (X-1, Y)),
move_south (north_force , (X, Y)) = ("north_force", (X, Y-1)),
move_north(south_force , (X, Y)) = ("south_force", (X, Y+1))
}

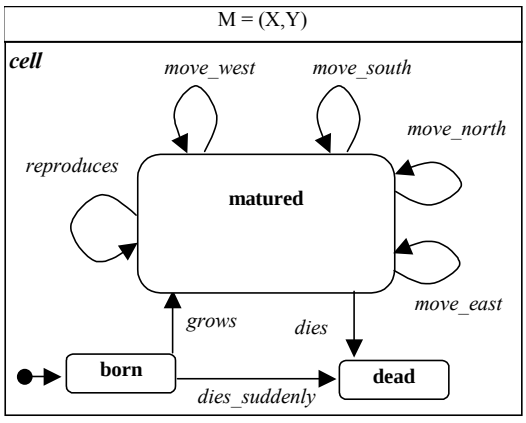

Figure 3: The next state transition function of a cell model.

Starting from the initial state q_0 (e.g. *born*) with the initial memory m_0 (e.g. *(16, 29)*), an input symbol $\sigma \in \Sigma$ (e.g. *mature_now*) triggers a function $\varphi \in \Phi$ (e.g. *grows*) which in turn causes a transition (from the set $F(q_0, \varphi)$) to a new state $q \in Q$ (e.g. *matured*) and a new memory value $m \in M$ (e.g. in this case the same *(16, 29)*). The sequence of transitions caused by the stream of input symbols is called a computation. The computation halts when all input symbols are consumed. The result of a computation is the sequence of outputs produced by the sequence of transitions.

Communicating X-machines

A Communicating X-machine model consists of several X-machines, which are able to exchange messages. These are normally viewed as inputs to some functions of an X-machine model, which in turn may affect the memory structure.

In principle, a *Communicating X-machine system* can be generally defined as a tuple $((C_i)_{i=1,\ldots,n}, R)$, where C_i is the ith X-machine component that participates in the system, and R is a communication relation between the n X-machines. There are several approaches in order to formally define a communicating X-machine. Some of them (Balanescu et al., 1999; Georgescu & Vertan, 2000) deviate from the original definition of the X-machine by introducing communicating functions as well as communicating states. As a result, developing a communicating component is perceived as a modelling activity different than that of developing a stand-alone X-machine. Consequently, one should start from scratch in order to specify a new component as part of the large system. In addition, components cannot be re-used as stand-alone X-machines or as components of other systems. Moreover, the semantics of the communicating functions impose a limited asynchronous operation of a communicating X-machine

Definition. In the current chapter, we define a *Communicating X-machine System Z* as a tuple, $Z=((C_1,\ldots,C_i,\ldots,C_n), CR)$, where:

- C_i is the ith X-machine component that participates in the system, and
- CR is a communication relation between the X-machine components, $CR \subseteq C \times C$, where $C = \{C_1, \ldots, C_n\}$. CR determines the communication channels that exist between the X-machines of the system. A tuple $(C_i, C_k) \in CR$ denotes that X-machine C_i can write

a message through a communication channel to a corresponding input stream of X-machine C_k for any $i, k \in \{1, \ldots, n\}, i \neq k$.

Let *alpha* be a function that returns the alphabet of an input or output stream of an X-machine. If is_i and os_i are the inputs and output streams of an X-machine X_i, then $alpha(is_i)=\Sigma_i$ and $alpha(os_i)=\Gamma_i$.

Definition. The *Communicating Input Streams* structure IS_i of an X-machine X_i is the a *n*-tuple $(is_{i1}, is_{i2}, \ldots, is_{ii}, \ldots, is_{in})$, that represents *n* input streams from which the machine X_i can receive messages sent by the other *n-1* X-machines, where:

- is_{ii} is the standard input stream of X_i
- $is_{ij} = \varepsilon$, if no communication is required with the jth X-machine $(alpha(\varepsilon) = \varnothing)$, or $alpha(is_{ij}) \subseteq \Sigma_i$.

Definition. A *Communicating Function-Input Stream Mapping* $\Phi IS_i \subseteq \Phi_i \times IS_i$ of an X-machine X_i is a relation between the functions of X_i and the streams they accept input from.

Definition. The *Communicating Output Streams* structure OS_i of an X-machine X_i is a *n*-tuple $(os_{i1}, os_{i2}, \ldots, os_{ii}, \ldots, os_{in})$ that represents *n* output streams to which the machine can send messages to be received by the other *n-1* X-machines, where:

- os_{ii} is the standard output stream of X_i
- $os_{ik} = \varepsilon$, if no communication is required with the kth X-machine $(alpha(\varepsilon) = \varnothing)$, or $alpha(os_{ik}) \subseteq \Sigma_k$.

Definition. A *Communicating Function-Output Stream Mapping* $\Phi OS_i \subseteq \Phi_i \times OS_i$ of an X-machine X_i is a relation between the functions of X_i and the streams they send their output to.

Definition. A *Communicating X-machine component* C_i is a 10-tuple:
$C_i = (\Sigma_i, \Gamma_i, Q_i, M_i, \Phi C_i, F_i, q_{0i}, m_{0i}, IS_i, OS_i)$
where ΦC_i is the new set of partial functions that read from either standard input or any other input stream and write to either the standard output or any other output stream.

More specifically, the set ΦC_i consists of four different sets of functions. The notation $(\sigma)_j$ used below denotes an incoming input from X-machine C_j while $(\gamma)_k$ denotes an outgoing output to X-machine C_k. The four different sets of functions are the following:

- the functions that read from standard input stream and write to standard output stream:

$\varphi_i(\sigma, m) = (\gamma, m')$

where $\sigma \in \Sigma_i$, $\gamma \in \Gamma_i$, $m,m' \in M_i$, $(\varphi_i \rightarrow is_{ii}) \in \Phi IS_i$ and $(\varphi_i \rightarrow os_{ii}) \in \Phi OS_i$.

- the functions that read from a communication input stream a message that is sent by another X-machine C_j and write to standard output stream:

$\varphi_i((\sigma)_j, m) = (\gamma, m')$

where $\sigma \in alpha(is_{ij})$, $\gamma \in \Gamma_i$, $m,m' \in M_i$, $(\varphi_i \rightarrow is_{ij}) \in \Phi IS_i$, $(\varphi_i \rightarrow os_{ii}) \in \Phi OS_i$, and $i \neq j$.

- the functions that read from standard input stream and write to a communication output stream a message that is sent to another X-machine C_k:

$\varphi_i(\sigma, m) = ((\gamma)_k, m')$

where $\sigma \in \Sigma_i$, $\gamma \in alpha(os_{ik})$, $m,m' \in M_i$, $(\varphi_i \rightarrow is_{ii}) \in \Phi IS_i$, $(\varphi_i \rightarrow os_{ik}) \in \Phi OS_i$ and $i \neq k$.

- the functions that read from a communication input stream a message that is sent by another X-machine C_j and write to a communication output stream a message that is sent to another X-machine C_k:

$\varphi_i((\sigma)_j, m) = ((\gamma)_k, m')$

where $\sigma \in alpha(is_{ij})$, $\gamma \in alpha(os_{ik})$, $m,m' \in M_i$, $(\varphi_i \rightarrow is_{ij}) \in \Phi IS_i$, $(\varphi_i \rightarrow os_{ik}) \in \Phi OS_i$ and $i \neq j$, $i \neq k$.

Instead of communicating with only one X-machine C_k, C_i may communicate with $C_{k1}, \ldots, C_{kp}$, sending them $\sigma_1, \ldots, \sigma_p$, respectively; in this case all (C_i, C_j), $1 \leq j \leq p$, must belong to CR and $(\gamma)_k$ will be replaced by $(\gamma_1)_{k1} \& \ldots \& (\gamma_p)_{kp}$.

Graphically, if a *solid circle* appears on a transition function, the function accepts input from a communicating stream instead of the standard input stream. It is read as "*reads from*". If a *solid diamond* appears on a transition function, this function may write to a communicating input stream of another X-machine. It is read as "*writes to*" (Figure 4).

Definition. A *Communication Interface* of an X-machine X_i is the tuple $(IS_i, OS_i, \Phi IS_i, \Phi OS_i)$ where:

- IS_i is the communicating input streams,
- OS_i is the communicating output streams,
- ΦIS_{ii} is the communicating Function-Input stream mapping.
- ΦOS_i is communicating Function-Output stream mapping.

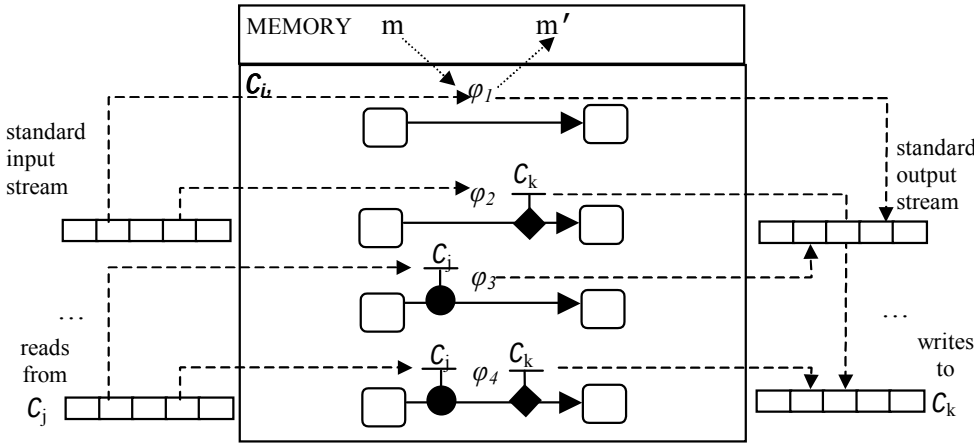

Figure 4: An abstract example of a communicating X-machine component with input and output streams and functions that receive input and produce output in any possible combination of sources and destinations.

Definition. Let X be the set of X-machines, CI the set of Communication Interfaces ($CI: IS \times OS \times FIS \times FOS$) and C the set of Communicating X-machine Components. The *Communication operator* is defined as:

Communication $COM: X \times CI \rightarrow C$

When the operator is applied, $COM((\Sigma_i, \Gamma_i, Q_i, M_i, \Phi_i, F_i, q_{0i}, m_{0i}), (IS_i, OS_i, \Phi IS_i, \Phi OS_i))$, it returns a communicating X-machine component C_i as a tuple: $C_i = (\Sigma_i, \Gamma_i, Q_i, M_i, \Phi C_i, F_i, q_{0i}, m_{0i}, IS_i, OS_i)$. A communicating component C_i contains in its tuple all the necessary compound constructs that integrate an X-machine and its communication interface $(IS_i, OS_i, \Phi IS_i, \Phi OS_i)$. These compound structures are made in such a way that separation to the sub-structures they are made of is possible. Therefore, it is feasible to apply the communication operator in reverse, that is, given a communicating component the reverse operation produces the X-machine 8-tuple and its communication interface:

Reverse Communication $COM^{-1}: C \rightarrow X \times CI$

Hereafter, the operator *COM* will be used with an infix notation (i.e. *x COM* y instead of *COM (x,y)*) and the operator COM^{-1} will be used with a prefix notation (i.e. COM^{-1}x instead of $COM^{-1}(x)$).

Adding a Biological Clock to a Cell Agent

Time is an important issue in modelling and simulating of biology-inspired systems. In cell systems, time plays the role of determining the life state of a cell (Walker et al., 2003). In P systems, computation is based in macro and micro cycles, which drive the evolution of individual membrane regions through the application of rewrite rules (Paun, 2002). Given the X-machine formalism, time can be trivially modeled in the memory structure and altered by clock inputs. New time values could then trigger functions that in turn change the state of the system. This solution assumes that an "*advance_time*" function will be attached to any state of the machine and a clock tick will be part of the input set, however, such an approach does not contribute to overall abstraction and simplicity. The X-machine model becomes rather complicated and includes functions that deal with time as well as functions that deal with the actual computation, e.g. the growth of a cell. We suggest that time can be handled separately by modelling a biological clock as an X-machine that communicates with the actual model of the cell.

In the previous example of the cell X-machine, the inputs that trigger the transitions are abstracted events that cause maturity, reproduction and death of a cell, in a form of explicit input symbols. However, the appropriate time to start the biological processes may be determined by the agent's biological clock, which can be modeled separately. For example, the biological clock is an X-machine of the form:

$Q = \{ticking\}$;
$\Sigma = \{tick\}$;
$\Gamma = \{mature\_now, reproduce\_now, die\_now\} \cup N_0$;
$M = (Clock, MatureAge, ReproductionAge, DieAge)$, where $MatureAge, ReproductionAge, DieAge \in N_0$;
$m_0 = (0,15, 40, 100)$, for example;
$q_0 = ticking$;

$\Phi = \{$

advance_time(tick, (Clock, MatureAge, ReproductionAge, DieAge)) =

(Clock+1, (Clock+1, MatureAge, ReproductionAge, DieAge))

if $Clock+1 \neq MatureAge \wedge Clock+1 \neq ReproductionAge \wedge Clock+1 \neq DieAge$,

time_to_grow(tick, (Clock, MatureAge, ReproductionAge, DieAge)) =

("mature_now", (Clock+1, MatureAge, ReproductionAge, DieAge))

if $Clock +1 = MatureAge$,

time_to_reproduce(tick, (Clock, MatureAge, ReproductionAge, DieAge)) =

("reproduce_now", (Clock+1, MatureAge, ReproductionAge, DieAge))

if $Clock+1 = ReproductionAge$,

time_to_die(tick, (Clock, MatureAge, ReproductionAge, DieAge)) =

("die_now", (Clock+1, MatureAge, ReproductionAge, DieAge))

if $Clock+1 = DieAge$

$\}$

The next state partial functions F of the biological clock (*bc*) and the X-machine previously described (*cell*) are shown in Figure 5. The two machines form a communicating system $((C_{cell}, C_{bc}), \{(C_{bc}, C_{cell})\})$. We call this communicating system an X^c-Machine (a clocked X-machine). The components of the X^c-machine are constructed through the application of the communication operator *COM* on *bc* and *cell* together with their respective communication interfaces as follows:

$C_{cell} = cell\ COM\ ((\Sigma^*_{cell}, is_{bc}), (\Gamma^*_{cell}, \varepsilon), \{grows \rightarrow is_{bc}, dies \rightarrow is_{bc}, reproduces \rightarrow is_{bc}\}, \{\ \})$

$C_{bc} = bc\ COM\ ((\varepsilon, \Sigma^*_{bc}), (os_{cell}, \Gamma^*_{bc}), \{\ \}, \{time\_to\_grow \rightarrow os_{cell}, time\_to\_die \rightarrow os_{cell}, time\_to\_reproduce \rightarrow os_{cell}\})$

This application will affect the corresponding functions of the individual components, for example *time_to_grow* and *grows* become:

time_to_grow(tick, (Clock, MatureAge, ReproduceAge, DieAge)) =

("mature_now" $_{cell}$*, (Clock+1, MatureAge, ReproduceAge, DieAge))*

if $Clock +1 = MatureAge$

grows(mature_now $_{bc}$ *, (X,Y)) =*

("agent matures", (X,Y))

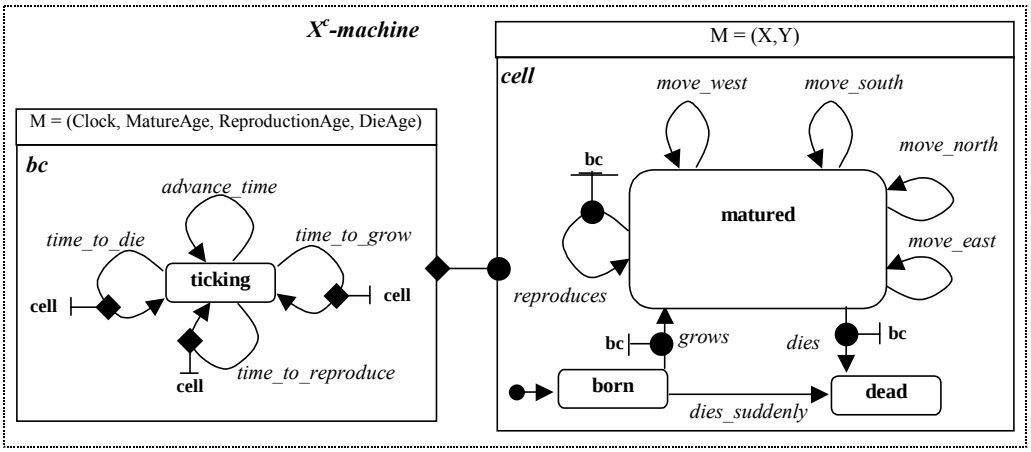

Figure 5**:** An X^c-machine consisting of the cell agent model and its biological clock.

A Tissue consisting of individual Cell Agents

So far, we have defined the part of the agent that deals with the agent life states depending on its age. A multi-agent system may be defined in the same way as a communicating X-machine system:

$Z = ((..., agent_i, ..., agent_j, ...), \{..., (agent_i, agent_j), ...\})$

where the first element is the tuple of the X^c-machines that participate in the system and the second element is the communication relation (Kefalas et al., 2003b). For example in Figure 1, initially the system is:

$Tissue = ((C_1, C_2, C_3, C_4, C_5), \{(C_1, C_2), (C_2, C_1), (C_1, C_3), (C_3, C_1), (C_1, C_4), (C_4, C_1)\})$

and it is shown as a communicating X-machine in Figure 6 (some states, the biological clocks and some functions are omitted for the sake of exposition). Communication is added to the *move* functions, for example in C_1 the function *move_south* becomes:

$move\_south(north\_force_{c2}, (X, Y)) = (\text{"}north\_force\text{"}_{c4}, (X, Y-1))$

which means that *move_south* reads a message from C_2 (*north_force*) and passes its output "*north_force*" to C_4. This will imply a change in the position of C_1 and accordingly a change in the position of C_4.

X-machines, as all other formal methods, provide a general framework based on a mathematical formalism that is able to describe what is intended. However, the resulting specification may still be based on over-simplifying assumptions. For example, in the above

model, the space is a two-dimensional one, the forces that move the cells can only come from four directions, move steps are discrete as imposed by the two-dimensional Cartesian space etc. If someone wishes to create a more detailed model then this is possible but requires extra effort and may result in a more complex model. For example, dealing with simultaneous forces that come from two different directions, requires the definition of new functions, which can deal explicitly with this situation. As it is defined above, the model deals implicitly with this situation without deadlocking, since the forces will impose movement of two cells in the same position at a particular instance, but in the long-term, while the system evolves (all inputs are consumed), the system will reach equilibrium and perform as intended.
Additionally to making the relevant assumptions and abstractions, the resulting specification or model may still contain errors, ambiguities or inconsistencies. These can be identified through techniques that are used for certifying model correctness as the ones described in the next section.

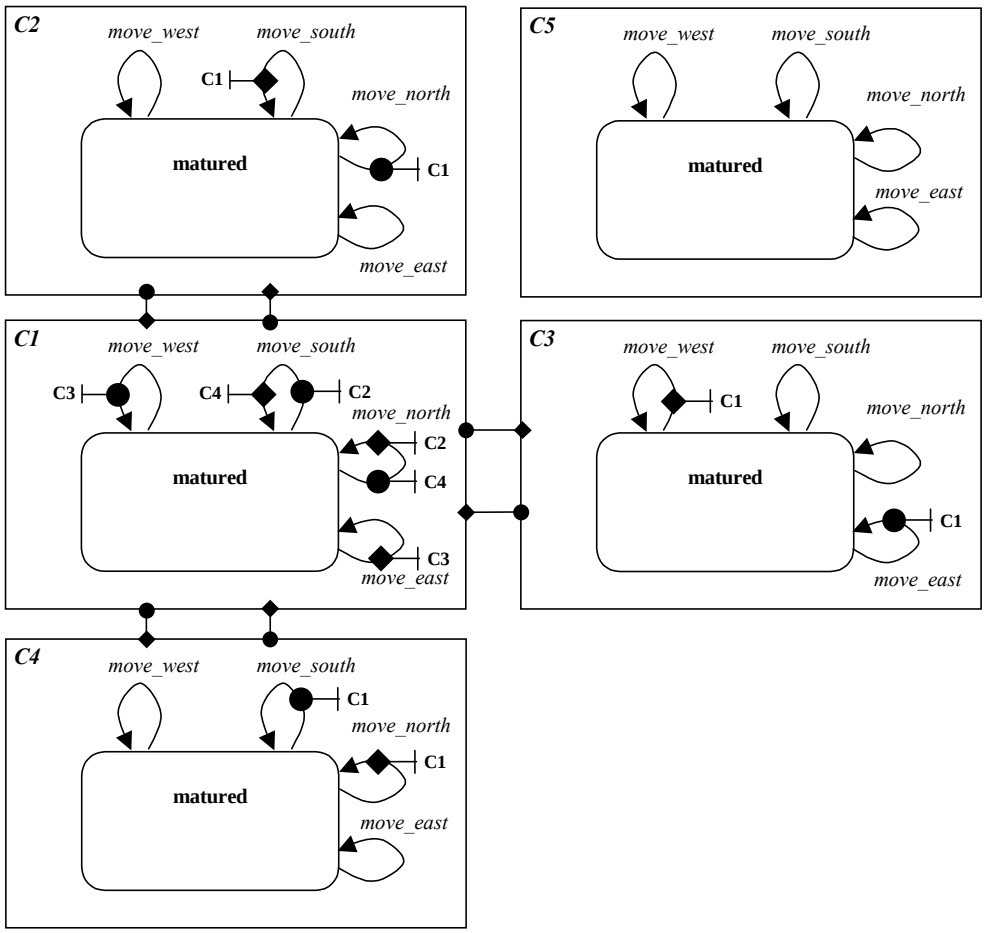

Figure 6: The tissue as a communicating X-machine system.

Modelling and Implementing a Correct System

The previous section demonstrated the usability and flexibility of the communicating X-machines as a modelling technique that incrementally develops a complex and complete model of a system. One may argue that the models are still too abstract while the assumptions made are over simplified. This is true to the extent that someone would like to construct a realistic system that would be simulated. However, we do not attempt something like this at present, something that could be extremely laborious as a task, but instead we try to demonstrate that the method is appropriate to model biology-inspired multi-agent systems. The resulting system would not need to be a simulation of the actual biological process (e.g. simulation of a colony of ants, growth of a cell tissue etc.), but a system or a technique that could be inspired by the biological process to solve various problems (e.g. ant colony optimization technique, artificial life agents etc.).

A significant advantage gained by an X-machine model is the ability to use formal verification and testing techniques. Verification and testing support each other and provide confidence in the correctness of the assumptions made. A formal verification technique for X-machines (model checking) is available, which determines whether desired properties of a system expressed in a variation of temporal logic, namely *XmCTL*, are satisfied by an X-machine model (Eleftherakis, 2003). Although it is not possible yet to formally verify the complete communicating model, it is possible to apply the model checking technique to the component X-machines (e.g. the biological clock *bc*). Thus, it is possible to verify all the components of a communicating system. For example, one property that someone might want to check against this model is that in all states the property *MatureAge<ReproductionAge<DieAge* is satisfied. This requirement can be expressed as an *XmCTL* formula as:

AGMx (M(2)<M(3) ∧ M(3)<M(4)),

which expresses that all memory instances (operator *Mx*) of all states (operators *A* and *G*) of the X-machine model satisfy that *MatureAge* is less than *ReproductionAge* which is less than *DieAge*, which are represented in the memory tuple in positions 2, 3 and 4 respectively (*M(2),M(3) and M(4)*).

There is a need for formal verification of communicating X-machine models as a whole and research is under way to extend the existing verification techniques also for communicating X-machine models. With a formal verification technique for the communicating model we will be able to express properties like: *it is impossible for two cells to occupy the same position at the same time*.

Having ensured that the model is "correct" with respect to the desired properties, we need to also ensure that the implementation is "correct", this time with respect to the model. This can be achieved through testing, but only under the important assumption that testing is complete. To guarantee "correctness" of the implementation, one must be certain that all tests are performed, and the results correspond to what the model has specified. As stated before, another strength of using stream X-machines to specify a system is that, under certain well defined conditions, it is possible to produce a test set that is guaranteed to determine the correctness of an implementation (Ipate & Holcombe, 1997; 1998). The test-set produced is proved to find all faults in the implementation. The testing process can therefore be performed automatically by checking whether the output sequences produced by the implementation are identical to the ones expected from the model.

DYNAMIC MODELLING

In the following, we are going to present a framework which is able to model the part of the system that is responsible for dynamically changing the configuration of the overall multi-agent system *Z*.

The kind of biology-inspired agent-based systems we would like to model and eventually simulate are highly dynamic, in the sense that new agents are created, some other cease to exist and communication channels change over time. It is therefore implied that in such dynamic systems the structure *Z*, which represents the communicating system, changes. A simplistic approach to tackle this problem is to allow a predefined number of models, arranged in some sort of a two or three dimensional space, to communicate with their neighbours and exchange messages that have as a result the turning on and off of their functionality. This happens, for example, in cellular automata, the most prominent example being the Conway's game of life and its variants (Gardner, 1970).

A more realistic approach would be to model agents (e.g. cells), which are part of biological systems, that are able to pass through a number of life states related to their age, such as being able to grow, reproduce, die etc. and at the same time to model the dynamic configuration of the multi-agent system (e.g. tissue), so that each system state has exactly the number of agents and communication channels that represent that particular state.

It is demonstrated in Kefalas et al. (2003a) how modelling of multi-agent systems can be done using the communicating X-machine approach. It is not shown, however, how the communicating X-machine system can alter its configuration when new agents enter or leave the system. The same kind of problem appeared in the attempt for simulating P systems using X-machines (Kefalas et al., 2003c), when a membrane structure alters its configuration by dissolving or dividing its component membranes. The solution provided is that a final "*dissolved*" state in each corresponding X-machine component and an explicit redirection of communicated objects through the dissolved component. This solution is, however, not very satisfactory. For the same reasons, membrane division was not addressed.

Summarizing the above issues, in order to model a dynamic biology-inspired system, the X-machine formalism should be able to handle effectively and neatly the following issues:

- Creation of new agent models that participate in the system,
- Deletion of agent models that cease to exist,
- Redirection of communication due to above reasons, but also due to other attributes of the system (e.g. physical movement through space that implies attachment or detachment of agents).

Since new agents are created while others are deleted from the system (e.g. cells are born while others die), X-machine models should be able to specify when and how this could take place. Instead of altering the definition of the X-machine, we suggest that this creation and deletion is handled separately from the agent model. Thus, the control that deals with the reconfiguration of the communicating system can be defined at a meta-level that may include special operators, capable of reconfiguring the system.

Changing a Communicating X-machine System

In this section, we are going to present four operators, which can change the configuration of a communicating X-machine system. The definition of only the first operator will be given. For the rest, examples of applying these operators to the cell system described above will be presented. The examples show how operators are applied step by step with specific parameters in order to produce a new communicating system.

There are four different operators that deal with reconfiguration of the communicating system, attachment *ATT*, detachment *DET*, generation *GEN*, and destruction *DES*. In the four definitions that follow ***P*** stands for powerset, *X* is the set of X-machines, *C* is the set of Communicating X-machine Components, *ΦIS* is the set of Function – Input Stream mappings, *ΦOS* is the set of Function – Output Stream mappings and *Z* the set of Communication X-machine Systems.

Definition. A communicating component may join a set of other components as a result of applying the operator *Attachment* defined as follows:

Attachment $ATT: (C \times \Phi IS \times \Phi OS) \times P(C \times \Phi IS \times \Phi OS) \times Z \rightarrow Z$

A communicating X-machine component C_i joins a set of other components $\{C_j \mid 1 \leq j \leq n \wedge i \neq j\}$. All components are associated with the functions that should read from and/or write to the corresponding input and/or output streams of other components. The result is a new set of communicating components having the appropriate communication channels between them.

The formal definition of the attachment operator is the following:

$$\forall (c_a, n\varphi is_a, n\varphi os_a): (C, \Phi IS, \Phi OS);\ \forall sc : P(C, \Phi IS, \Phi OS);\ \forall (cx, cr): Z \bullet$$
$$ATT((c_a, n\varphi is_a, n\varphi os_a), sc, (cx, cr)) = (cx', cr')$$

where:

$$cx' = (cx \setminus (\{c_a\} \cup \{c_b \mid (c_b, n\varphi is_b, n\varphi os_b) \in sc\})) \cup \{c_a'\} \cup nc$$

$$cr' = cr \cup \{(c_a, c_b) \mid (\varphi, is_a) \in n\varphi is_b\} \cup \{(c_b, c_a) \mid (\varphi, is_b) \in n\varphi is_a\}$$

and

$$COM^{-1}\ c_a = (x_a, (IS_a, OS_a, \varphi is_a, \varphi os_a))$$

$c_a' = x_a\ COM\ (IS_a \otimes_{t1} T_{1a},\ OS_a \otimes_{t2} T_{2a},\ \varphi is_a \cup n\varphi is_a,\ \varphi os_a \cup n\varphi os_a)$

$nc = \{\ c_b' \mid$

$COM^{-1}\ c_b = (x_b, (IS_b, OS_b, \varphi is_b, \varphi os_b)) \wedge$

$c_b' = x_b\ COM\ (IS_b \otimes_{<a>} T_{1b},\ OS_b \otimes_{<a>} T_{1b},\ \varphi is_b \cup n\varphi is_b,\ \varphi os_b \cup n\varphi os_b)$

$\wedge$

$(c_b, n\varphi is_b, n\varphi os_b) \in sc\}$

In the above, the ordered sequences $t_1, t_2, T_{1a}, T_{2a}, T_{1b}, T_{2b}$ are defined as:

$t_1 = ordered <i \mid is_i \in ran(n\varphi is_a)>$

$t_2 = ordered <i \mid os_i \in ran(n\varphi os_a)>$

$T_{1a} = ordered <is_i \mid is_i \in ran(n\varphi is_a)>$

$T_{2a} = ordered <os_i \mid os_i \in ran(n\varphi os_a)>$

$T_{1b} = ordered <is_i \mid is_i \in ran(n\varphi is_b)>$

$T_{2b} = ordered <os_i \mid os_i \in ran(n\varphi os_b)>$

where *ordered* is a function that returns the ordered set (sequence) and *ran* denotes the range of a relation set. The operator $\otimes_{<\ldots,i,\ldots>}$ overrides the i^{th} element of the tuple, for example: (a, b, c) $\otimes_{<1,3>}$ <z, y> = (z, b, y).

Consider Figure 1 where C_5 joins the cluster of cells to the left (C_5 is attached to C_2 and C_3). The operation is defined as:

$ATT ((C_5, N\Phi IS_5, N\Phi OS_5), \{(C_2, N\Phi IS_2, N\Phi OS_2), (C_3, N\Phi IS_3, N\Phi OS_3)\}, Tissue) = Tissue'$

where:

$N\Phi IS_5 = \{(move\_east, is_2), (move\_north, is_3)\}$

$N\Phi OS_5 = \{(move\_west, os_2), (move\_south, os_3)\}$

$N\Phi IS_2 = \{(move\_west, is_5)\}$

$N\Phi IS_3 = \{(move\_south, is_5)\}$

$N\Phi OS_2 = \{(move\_east, os_5)\}$

$N\Phi OS_3 = \{(move\_north, os_5)\}$

Algorithmically, one can go through the steps of applying the operator one by one as follows:

1: $COM^{-1}\ C_5 = (X_5, (IS_5, OS_5, \Phi IS_5, \Phi OS_5))$

destruct the communicating component C_5 into its X-machine X_5 and its communication interface.

2: $C_5' = X_5\ COM\ (IS_5 \otimes_{<2,3>} <is_2,is_3>,\ OS_5 \otimes_{<2,3>} <os_2,os_3>,\ \Phi IS_5 \cup N\Phi IS_5,\ \Phi OS_5 \cup N\Phi OS_5$

construct the new communicating component C_5' by using the X_5 machine together with its new communication interface.

3: $COM^{-1}\ C_2 = (X_2, (IS_2, OS_2, \Phi IS_2, \Phi OS_2))$

destruct the communicating component C_2 into its X-machine X_2 and its communication interface.

4: $C_2' = X_2\ COM\ (IS_2 \otimes_{<5>} <is_5>,\ OS_2 \otimes_{<5>} <os_5>,\ \Phi IS_2 \cup N\Phi IS_2,\ \Phi OS_2 \cup N\Phi OS_2)$

construct the new communicating component C_2' by using the X_2 machine together with its new communication interface.

5: $COM^{-1}\ C_3 = (X_3,\ (IS_3, OS_3, \Phi IS_3, \Phi OS_3)\ )$

destruct the communicating component C_3 into its X-machine X_3 and its communication interface.

6: $C_3' = X_3\ COM\ (IS_3 \otimes_{<5>} <is_5>,\ OS_3 \otimes_{<5>} <os_5>,\ \Phi IS_3 \cup N\Phi IS_3,\ \Phi OS_3 \cup N\Phi OS_3)$

construct the new communicating component C_3' by using the X_3 machine together with its new communication interface.

7: $Tissue' = ((C_1, C_2', C_3', C_4, C_5'), \{(C_1, C_2'), (C_2', C_1), (C_1, C_3'), (C_3', C_1), (C_1, C_4), (C_4, C_1), (C_2', C_5'), (C_5', C_2'), (C_3', C_5'), (C_5', C_3')\})$

Construct the new tissue by changing the old communicating components C_2, C_3, C_5 with the newly constructed ones C_2', C_3', C_5' and add the new tuples $(C_2', C_5'), (C_5', C_2'), (C_3', C_5'), (C_5', C_3')$ in the communication relation.

Definition. A communicating component may depart from a set of other components as a result of applying the operator *Detachment* defined as follows:

Detachment $\boldsymbol{DET: C \times P\ C \times Z \rightarrow Z}$

A communicating X-machine component C_i is detached from a set of other components $\{C_j \mid 1 \leq j \leq n \wedge i \neq j\}$. The result is a new set of communicating components having the appropriate communication channels between them (i.e. the relevant functions that used to read from and/or write to the detached component are removed). The latter applies to the detached component as well.

For example, consider Figure 1 where C_4 leaves the cluster of cells (C_4 is detached from C_1). The operation is defined as:

$DET(\ C_4, \{C_1\}, Tissue) = Tissue'$

Algorithmically, the step-by-step application of the operator in the above expression is as follows:

1: $COM^1\ C_4\ = (X_4, (IS_4, OS_4, \Phi IS_4, \Phi OS_4))$

2: $C_4' = X_4\ COM\ (IS_4 \otimes_1 \varepsilon, OS_4 \otimes_1 \varepsilon, \Phi IS_4 \backslash \{\varphi_i \rightarrow is_1\}, \Phi OS_4 \backslash \{(\varphi_i \rightarrow os_1)\})$

3: $COM^1\ C_1 = (X_1, (IS_1, OS_1, \Phi IS_1, \Phi OS_1))$

4: $C_1' = X_1\ COM\ (IS_1 \otimes_4 \varepsilon, OS_1 \otimes_4 \varepsilon, \Phi IS_1 \backslash \{\varphi_i \rightarrow is_4\}, \Phi OS_1 \backslash \{\varphi_i \rightarrow os_4\})$

5: $Tissue' = ((C_1', C_2, C_3, C_4', C_5), \{(C_1', C_2), (C_2, C_1'), (C_1', C_3), (C_3, C_1'), (C_2, C_5), (C_5, C_2), (C_3, C_5), (C_5, C_3)\})$

where \ denotes the set difference.

Definition. A newly constructed X-machine model may join a communicating system as a result of applying the operator *Generation* defined as follows:

Generation $GEN: (X \times \Phi IS \times \Phi OS) \times P\ (C \times \Phi IS \times \Phi OS) \times Z \rightarrow Z$

A new communicating X-machine component C_i is created from X, which acts as a genetic code for the new component, and joins a set of other components $\{C_j \mid 1 \leq j \leq n \wedge i \neq j\}$. All components, new and old ones, are associated with the functions that should read from and/or write to the corresponding input and/or output streams of other components. The result is a new set of communicating components having the appropriate communication channels between them. The rest of the components that are not affected are informed about the possibility of having a communication channel open with the new component, by having a new element added to their communicating input and output streams.

Consider Figure 1 where C_6 is born and joins the cluster of cells (C_6 is attached to C_1 and C_4). The operation is defined as:

$GEN\ (\ (X_6, N\Phi IS_6, N\Phi OS_6), \{(C_1, N\Phi IS_1, N\Phi OS_1), (C_4, N\Phi IS_4, N\Phi OS_4)\}, Tissue) = Tissue'$

where:

$N\Phi IS_6 = \{(move\_south, is_1), (move\_north, is_4)\}$

$N\Phi OS_6 = \{(move\_north, os_1), (move\_south, os_4)\}$

$N\Phi IS_1 = \{(move\_north, is_6)\}$

$N\Phi IS_4 = \{(move\_south, is_6)\}$

$N\Phi OS_1 = \{(move\_south, os_6)\}$

$N\Phi OS_4 = \{(move\_north, os_6)\}$

Algorithmically, the step-by-step application of the operator in the above expression is as follows:

1: $C_6' = X_6\ COM\ ((is_1, \varepsilon, \varepsilon, is_4, \varepsilon, \Sigma_6^*), (os_1, \varepsilon, \varepsilon, os_4, \varepsilon, \Gamma_6^*), N\Phi IS_6, N\Phi OS_6)$

2: $COM^{-1}\ C_1 = (X_1, (IS_1, OS_1, \Phi IS_1, \Phi OS_1))$

3: $C_1' = X_1\ COM\ (IS_1 \oplus <is_6>, OS_1 \oplus <os_6>, \Phi IS_1 \cup N\Phi IS_1, \Phi OS_1 \cup N\Phi OS_1)$

4: $COM^{-1}\ C_4 = (X_4, (IS_4, OS_4, \Phi IS_4, \Phi OS_4))$

5: $C_4' = X_4\ COM\ (IS_4 \oplus <is_6>, OS_4 \oplus <os_6>, \Phi IS_4 \cup N\Phi IS_4, \Phi OS_4 \cup N\Phi OS_4)$

6: $COM^{-1}\ C_i = (X_i, (IS_i, OS_i, \Phi IS_i, \Phi OS_i)),\ i \neq 1, 4, 6$

7: $C_i' = X_i\ COM\ (IS_i \oplus <\varepsilon>, OS_i \oplus <\varepsilon>, \Phi IS_i, \Phi OS_i),\ i \neq 1, 4, 6$

8: $Tissue' = ((C_1', C_2', C_3', C_4', C_5', C_6'), \{(C_1', C_2'), (C_2', C_1'), (C_1', C_3'), (C_3', C_1'), (C_1', C_6'), (C_6', C_1'), (C_2', C_5'), (C_5', C_2'), (C_3', C_5'), (C_5', C_3'), (C_4', C_6'), (C_6', C_4')\})$

The operator $\oplus$ adds elements at the end of a tuple, for example: $(a, b, c) \oplus <x, y> = (a, b, c, x, y)$

Definition. A communicating component may cease to exist in a communicating system as a result of applying the operator *Destruction* defined as follows:

Destruction $\boldsymbol{DES: C \times P\ C \times Z \rightarrow Z}$

An existing communicating X-machine component C_i ceases to exist and is detached from a set of other components $\{C_j \mid 1 \leq j \leq n \wedge i \neq j\}$. The result is a new set of communicating components having the appropriate communication channels between them (i.e. the relevant functions that used to read from and/or write to the removed component are removed).

Consider Figure 1 where C_1 dies (C_1 is removed and detached from C_2, C_3 and C_6). The operation is defined as:

$DES\ (C_1, \{C_2, C_3, C_6\}, Tissue) = Tissue'$

Algorithmically, the step-by-step application of the operator in the above expression is as follows:

1: $COM^{-1}\ C_2 = (X_2, (IS_2, OS_2, \Phi IS_2, \Phi OS_2))$

2: $C_2' = X_2\ COM\ (IS_2 \otimes_{<1>} <\varepsilon>, OS_2 \otimes_{<1>} <\varepsilon>, \Phi IS_2 \backslash \{\varphi_i \rightarrow is_1\}, \Phi OS_2 \backslash \{\varphi_i \rightarrow os_1\})$

3: $COM^{-1}\ C_3 = (X_3, (IS_3, OS_3, \Phi IS_3, \Phi OS_3))$

4: $C_3' = X_3\ COM\ (IS_3 \otimes_{<1>} <\varepsilon>, OS_3 \otimes_{<1>} <\varepsilon>, \Phi IS_3 \backslash \{\varphi_i \rightarrow is_1\}, \Phi OS_3 \backslash \{\varphi_i \rightarrow os_1\})$

5: $COM^{-1}\ C_6 = (X_6, (IS_6, OS_6, \Phi IS_6, \Phi OS_6))$

6: $C_6' = X_6\ COM\ (IS_6 \otimes_{<1>} <\varepsilon>, OS_6 \otimes_{<1>} <\varepsilon>, \Phi IS_6 \backslash \{\varphi_i \rightarrow is_1\}, \Phi OS_6 \backslash \{\varphi_i \rightarrow os_1\})$

7: $Tissue' = (\{C_2', C_3', C_4, C_5, C_6'\}, \{(C_2', C_5), (C_5, C_2'), (C_3', C_5), (C_5, C_3'), (C_4, C_6'), (C_6', C_4)\})$

As stated above, our intention is to create a control at a meta-level, which having these special operations will be able to reconfigure the communicating system. There are mainly two ways to achieve that:

- Control is global to the whole communicating system.
- Control is attached locally to each agent.

These two ways will be investigated in the following sections. One can imagine a middle approach (by which control stands between agents that communicate, like a bond) but its analysis does not fall under the scope of this chapter.

Modelling a Dynamic System: The Global Approach

Consider a multi-agent system model created by developing a communicating X-machine Z, using X-machine components C_i that communicate directly using the approach described above. In this first approach, the information that drives the dynamic evolution of the system is global and can be described at a meta-level as:

- a set of meta-rules (i.e. rules that refer to the configuration of the system), or
- a meta-machine (i.e. an X-machine-like device that manipulates the components or attributes of the system).

Conceptually, the actual communicating system and the meta-system could be considered to play the role of the environment to each other. For example, the global control receives input from its environment, in this case *Z*, processes this input and returns an output in form of operations described above, which in turn change the environment (Figure 7).

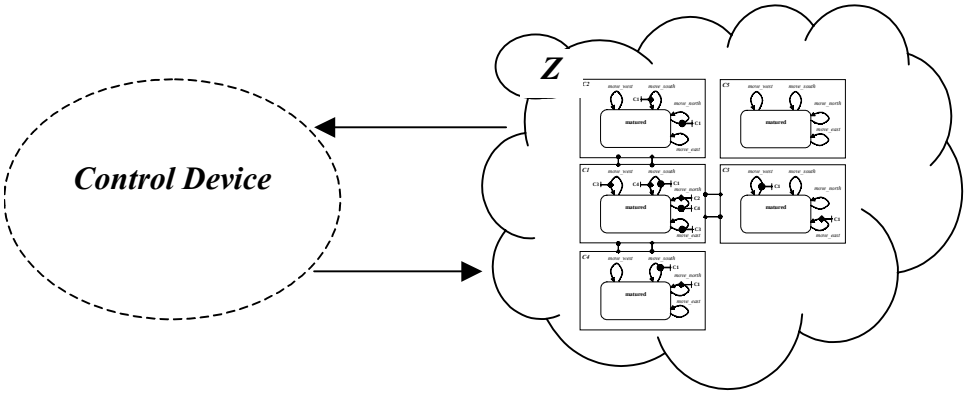

Figure 7: Global approach for modelling a dynamic system.

The control device should be able to control the change of the configuration of the communicating system, and therefore should possess the following information at any time:

- The tuple that represents the communicating system $Z = ((C_1, ..., C_i, ..., C_n), CR)$
- The current system state (*SZ*) of *Z*. *SZ* is defined as a set of tuples $SZ=\{... (q_c, M_c, \varphi_c)_i, ...\}$ with each tuple corresponding to a component C_i: the current state (q_c) in which C_i is at, the current memory (M_c) of C_i and the last function (φ_c) that was applied in C_i,
- Definitions of all components that exist or may be added to the system. These definitions act as genetic codes (*GC*) for the system. *GC* is a set of tuples, $GC = \{... (\Sigma, \Gamma, Q, M, \Phi, F, \Phi_R, \Phi_W)_j, ...\}$ where the first six elements are as defined in the definition of the X-machine given in the previous section and the last two are the set of functions that may be involved in communication with other components (i.e. Φ_R may read from communicating streams and Φ_W may write to communicating streams).

Using the above information, the control device can perform the following actions:

- generate a new component and attach it to the communicating machine *Z*,
- destruct an existing component of *Z* and rearrange the communication of the rest components appropriately, and

- change the communication between a group of components because of the addition or removal of an existing component to or from the communicating cluster (attachment, detachment).

All these are possible through the use of the operators that were presented in the previous section. For example in Figure 1, consider just the first case, where the cell C_5 is attached to the group of cells because it receives a force from the east that has as a result for C_5 to move to the west. The move implies that C_5 has to initiate communication with C_2 and C_3, since in this case the criterion for establishing communication is the immediate neighbourhood. The following is a step-by-step description of how this is possible.

The state of the communicating system *SZ* is updated because of the movement of C_5. The control device identifies the attachment of this component to the other two by comparing the coordinates of this cell with the coordinates of the rest of the cells (all represented as components of *Z*). This is possible because all the required information is available through the memory instances of all the components in *SZ*. Comparing the coordinates the control device identifies that C_5 is now next to C_2 and C_3, and therefore it should apply the appropriate operators that change Z to a new, evolved, communicating system *Z'*, in this case by just altering the communication relation *CR*. For this example the attachment operator should be used as follows:

$ATT ((C_5, N\Phi IS_5, N\Phi OS_5), \{(C_2, N\Phi IS_2, N\Phi OS_2), (C_3, N\Phi IS_3, N\Phi OS_3)\}, Z) = Z'$

To construct $N\Phi IS_i$ the genetic code of the component representing the cell C_i should be known, information available to the control machine by the set *GC*. For example for C_5, the genetic code provides the information that the set of the functions that may read from a communicating stream is $\Phi_{R5} = \{move\_east, move\_west, move\_south, move\_north\}$ and the set of functions that may write to a communicating stream is $\Phi_{W5} = \{move\_east, move\_west, move\_south, move\_north\}$. Reasoning about the particular system and taking into account the particular rules of communication, it is now possible to create $N\Phi IS_5 = \{(move\_east \rightarrow is_2), (move\_north \rightarrow is_3)\}$ and $N\Phi OS_5 = \{(move\_west \rightarrow os_2), (move\_south \rightarrow os_3)\}$. The same reasoning is applied to construct $N\Phi IS_2$, $N\Phi OS_2$, $N\Phi IS_3$, and $N\Phi OS_3$.

Modelling a Dynamic System: The Local Approach

The second approach supports a local view for control, meaning that each component of the communicating X-machine system Z is wrapped around an interface device. This is a specialised communicating device that is able to communicate with all other interfaces. As above, the device is defined at a meta-level for each component either as a set of meta-rules or as a meta-machine.

The interface should be able to control the change of the configuration of the communicating system, and therefore should possess information similar to the global control but this time having only the local view of the component C_i that this interface controls:

- The tuple that represents part of the communicating system $Z_p = ((C_1, \ldots, C_i, \ldots, C_n), CR_p)$, (i.e. not all components but only the subset CR_p of CR that refers to the communication of C_i).
- The current component state (SC_i) of C_i. SC_i is defined as a tuple $SC_i = (q_c, M_c, \varphi_c)_i$, with each element of the tuple representing: the current state (q_c) in which C_i is at, the current memory (M_c) of C_i and the last function (φ_c) that was applied in C_i,
- The genetic codes (GC_i) for SC_i. GC_i is a tuple, $GC_i = (\Sigma, \Gamma, Q, M, \Phi, F, \Phi_R, \Phi_W)_j$, as before.

Communication between components of the system is only possible through their interfaces. Therefore, we have an explicit one-to-all communication between interfaces and implicit one-to-many communication between components (Figure 8). An interface broadcasts control messages that are received by all others, but it can also broadcast messages coming from its local component C_i that are received only by the interfaces whose components are supposed to communicate with C_i. As before, the interface applies operators to the system and changes it. The only difference with the global control is that now information is only locally available and whenever extra information is needed, it can be retrieved by exchanging control messages with the other interfaces.

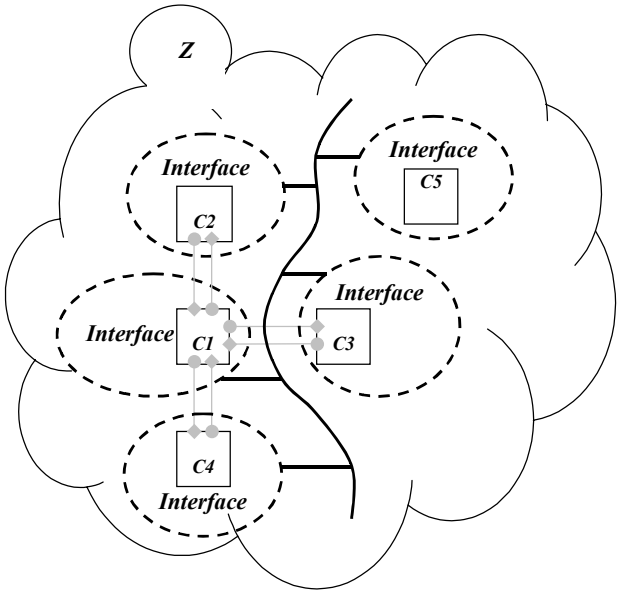

Figure 8: Local approach for modelling a dynamic system.

DISCUSSION

The approach described in this chapter aims towards the development of formal models of biology-inspired agents. There are two fundamental concepts associated with any dynamic or reactive system, such as an agent, that is situated in some environment and reacting with it. Firstly, it is the environment itself, which could be precisely or ill-specified or even completely unknown and involves identifying its important aspects and the way in which they may change in accordance with the activities of the agent. Secondly, it is the agent, which responds 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: it undergoes internal changes and it produces outputs that affect the environment.

The above concepts are captured by X-machines as demonstrated by the example presented. X-machine models can be created as stand-alone agent models. These models can be verified through XmCTL and model checking algorithms and tested, so that their "correctness" is proven. It is then possible to construct a communicating agent model through the use of communicating streams. The important issue addressed is that the communicating components can be derived from stand-alone models when combined with their communication interface. This has a tremendous practical advantage in the sense that the tools constructed for X-machines can incrementally develop communicating systems without

destroying stand-alone models (Kefalas et al., 2003b). The X-machine method is rather intuitive, while formal descriptions of data types and functions can be expressed in any known mathematical notation. However, X-machine modelling tools support a specific notation, namely XMDL, which provides all the necessary constructs for mathematical modelling through simple text (Kefalas & Kapeti, 2000).

We have also defined a set of operators that are able to reconfigure a communicating agent system. These operators are going to be useful when modelling of the control or interface devices is attempted, either at a global or at a local level. Actually, the operators are actions of the control or interface devices through which the evolution of the communicating system takes place. Defining such devices or interfaces, either in the form of meta-rules or meta-machines, is rather complex and specific to the problem at hand. This is because modelling of such a device in the above described example falls out of the scope of this chapter. One should have in mind that the global and the local approaches have their usual advantages and disadvantages, that is, completeness and availability of information versus communication overhead.

However, bearing in mind that the model should lead towards the implementation of such systems, the implementer may choose the most appropriate technique for modelling the control or interface device, or even to combine both approaches into a hybrid one.

CONCLUSIONS

In this chapter we have showed how modelling of multi-agent systems can be achieved through the use of communicating X-machines and how four meta-operators can change the configuration of the system. By taking as an example a biology-inspired multi-agent system, we demonstrated that control over the global system state should be modelled in a way that does not affect modelling of individual components of the system. The important contribution of this work is that overall modelling of multi-agent systems can be achieved by:

- Decoupling modelling of agents and modelling of time by using the communicating X-machine approach, and

- Decoupling modelling of agents and multi-agent reconfiguration by using operators at a meta-level, which will be responsible for dynamically configuring the system as time progresses.

This allows a modular and disciplined approach towards modelling. Modular, since the components of a multi-agent system can be modeled separately. By having minimal changes in the X-machine model we are still able to exploit to the maximum possible extent the legacy of X-machine theory concerning complete testing and formal verification for each individual component. Such models may comprise a library of off-the-shelf, ready-made, verified and tested components, which may be re-used in other systems. Disciplined, since the modelling activity can be performed in stages. First, individual models are developed and then the whole multi-agent system is constructed. From a practical point of view, this creates a significant advantage to the modeler.

Future work involves the development of a theory for verifying and testing communicating systems as a whole. It is also worth investigating how complex it is to model actual biological processes through formal methods and what are the benefits of such modelling. Finally, the tools for X-machines, apart from animation of stand-alone components, need to be equipped with animation of whole communicating systems.

REFERENCES

A Bibliography of Molecular Computation and Splicing Systems (2003). http://www.wi.leidenuniv.nl/~pier/dna.html

Adleman, L.M. (1994) Molecular computation of solutions to combinatorial problems. Science, 226, 1021 - 1024.

Attoui, A., & Hasbani, A. (1997) Reactive systems developing by formal specification transformations. Proceedings of the 8th International Workshop on Database and Expert Systems Applications (DEXA 97), 339 - 344.

Balanescu, T., Gheorghe, M., Holcombe, M., & Ipate, F. (2002) A variant of EP systems. Pre-proc. Workshop on Membrane Computing, Curtea de Arges, Romania, Publication No.1, MolCoNet project – IST – 2001 – 32008, 57 – 64.

Balaneascu, T., Cowling, A.J., Georgescu, H., Gheorghe, M., Holcombe, M., & Vertan, C. (1999) Communicating Stream X-machines Systems are no more than X-machines. Journal of Universal Computer Science. 5 (9), 494 - 507.

Banatre, J-P., & Le Metayer, D. (1990) The gamma model and its discipline of programming. Science of Computer Programming. 15, 55 - 77.

Benerecetti, M., Giunchiglia, F., & Serafini, L. (1999) A model-checking algorithm for multi-agent systems. In (Muller, J. P., M. P. Singh, M. P., & Rao, A. S., eds. (1999) Intelligent Agents V, Lecture Notes in Artificial Intelligence, Berlin: Springer), 163 - 176.

Berry, G., & Boudol, G. (1992) The chemical abstract machine. Theoretical Computer Science. 96, 217 - 248.

Brazier, F., Dunin-Keplicz, B., Jennings, N., & Treur, J. (1995) Formal specification of multi-agent systems: a real-world case. Proceedings of International Conference on Multi-Agent Systems (ICMAS'95), MIT Press, 25 - 32.

D'Inverno, M., Kinny, D., Luck, M., & Wooldridge, M. (1998) A formal specification of dMARS. In (Singh, M. P., Rao, A., & Wooldridge, M. J., eds. (1998) Intelligent Agents IV, Lecture Notes in Artificial Intelligence, 1365, Berlin: Springer), 155 - 176.

Duan, Z. H., Holcombe, M., & Linkens, D. A. (1995) Modelling of a soaking pit furnace in hybrid machines. System Analysis, Modelling, Simulation. 18, 153 - 157.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996) The Ant System: Optimisation by a colony of co-operating agents. IEEE Transactions on Systems, Man and Cybernetics. 26(1), 1 – 13.

Eilenberg, S. (1974) Automata, Languages and Machines, vol. A, Academic Press.

Eleftherakis, G. (2003) Formal Verification of X-machines Models: Towards Formal Development of Computer-Based Systems. PhD Thesis, University of Sheffield, Department of Computer Science.

Eleftherakis, G., & Kefalas, P. (2001) Model checking safety critical systems specified as X-machines. Analele Universitatii Bucharest, Mathematics-Informatics series. 49(1), 59 - 70.

Fisher, M., & Wooldridge, M. (1997) On the formal specification and verification of multi-agent systems. International Journal of Cooperating Information Systems. 6(1), 37 - 65.

Futatsugi, K., Goguen, J., Jouannaud, J.P., & Meseguer, J. (1985) Principles of OBJ2. In (B. Reid, ed. (1985) Proceedings of Twelfth ACM Symposium on Principles of Programming Languages, Association for Computing Machinery), 52 - 66.

Gardner, M. (1970) Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life". Scientific American. 223, 120 - 123.

Gheorghe, M., Holcombe, M., & Kefalas, P. (2001) Computational models of collective foraging. BioSystems. 61, 133 - 141.

Georgescu, H., & Vertan, C. (2000) A new approach to Communicating X-machines systems. Journal of Universal Computer Science. 6(5), 490 - 502.

Gheorghe, M., Holcombe, M., & Kefalas, P. (2003) Eilenberg P systems: A bio-computational model. In Proceedings of the 1st Balkan Conference in Informatics, November 2003, 147 - 160.

Harel, D. (1987) Statecharts: A visual approach to complex systems. Science of Computer Programming, 8(3), 231–274

Holcombe, M. (1988) X-machines as a basis for dynamic system specification. Software Engineering Journal. 3(2), 69 - 76.

Holcombe, M., & Ipate, F. (1998) Correct Systems: Building a Business Process Solution. Springer-Verlag, London.

Holcombe, M. (2001) Computational models of cells and tissues: Machines, agents and fungal infection. Briefings in Bioinformatics. 2, 271 - 278.

Ipate, F., & Holcombe, M. (1997) An integration testing method that is proved to find all faults. International Journal of Computer Mathematics. 63(3), 159 - 178.

Ipate, F., & Holcombe, M. (1998) Specification and testing using generalised machines: A presentation and a case study. Software Testing, Verification and Reliability. 8, 61 - 81.

Jennings, N.R. (2000) On agent-based software engineering. Artificial Intelligence. 117, 277 - 296.

Jones, C. B. (1990) Systematic Software Development using VDM (2nd ed.). Prentice-Hall.

Kauffman, S.A. (1993) The Origins of Order - Self-organization and Selection in Evolution, Oxford University Press.

Kefalas P. (2002) Formal modelling of reactive agents as an aggregation of simple behaviours. In (Vlahavas, I. P., & Spyropoulos, C. D., eds. (2002) Lecture Notes in Artificial Intelligence 2308, Berlin: Springer), 461 - 472.

Kefalas, P., Holcombe, M., Eleftherakis, G., Gheorghe, M. (2003a) A formal method for the development of agent based system. In (Plekhavona, V. ed., (2003) Intelligent Agent Software Engineering, Idea Group Publishing), 68 - 98.

Kefalas, P., Eleftherakis, G., & Kehris, E. (2003b) Communicating X-machines: A practical approach for formal and modular specification of large systems. Journal of Information and Software Technology. 45(5), 269 - 280.

Kefalas P., Eleftherakis, G., Holcombe, M., & Gheorghe, M. (2003c) Simulation and verification of P systems through communicating X-machines. Biosystems. 70(2), 135 - 148.

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

Odell, J., Parunak, H. V. D., & Bauer, B. (2000) Extending UML for agents. In Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, 3-17.

Păun, Gh. (2000) Computing with membranes. Journal of Computer and System Sciences. 61(1), 108 - 143.

Păun, Gh. (2002) Membrane Computing: An Introduction. Berlin: Springer.

Rao, A. S., & Georgeff, M. P. (1993) A model-theoretic approach to the verification of situated reasoning systems. In (Bajcsy, R., ed., (1993) Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93). Morgan Kaufmann), 318 - 324.

Rao, A. S., & Georgeff, M. P. (1995) BDI Agents: from theory to practice. In Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95), 312 - 319.

Reisig, W. (1985) Petri nets - an introduction. EATCS Monographs on Theoretical Computer Science, 4, Berlin: Springer.

Rosenschein, S. R., & Kaebling, L. P. (1995) A situated view of representation and control. Artificial Intelligence. 73(1-2), 149 - 173.

Spivey, M. (1989) The Z Notation: A Reference Manual. Prentice-Hall.

The P Systems Web Page (2003). http://psystems.disco.unimib.it

Walker, D. C., Holcombe, M., Southgate, Mac Neil, S., J. & Smallwood, R. H. (2004) Agent-based modeling of the social behaviour of cells. Biosystems. (To appear)

Wulf, W. A., Shaw, M., Hilfinger, P. N., & Flon, L. (1981) Fundamental Structures of Computer Science. Addison-Wesley.