Using X-Machines as a Formal Basis for Describing Agents in Agent-Based Modelling

Simon Coakley, Rod Smallwood and Mike Holcombe
University of Sheffield
North Campus, Broad Lane, Sheffield, S3 7HQ, UK
{S.Coakley,R.Smallwood,M.Holcombe}@dcs.shef.ac.uk

Keywords: X-machines, systems biology, agent-based, formal, architecture

Abstract

This paper overviews a formal agent-based modelling architecture that has been designed around the need for multi-agent modelling of biological cells with respect to inter-cellular signalling and tissue, and intra-cellular pathways. The specific aim is to provide a way to formally describe individual components of a biological system as agents using a generic framework that is applicable to any system. This paper proposes the formalisation of agents as communicating computing machines, based on communicating stream X-machines. The inherent parallel ability of the architecture is described by the use of local message lists and models are described that are currently in development using the architecture.

1. INTRODUCTION

With the increasing use of agent-based modelling in biology, modelling cellular interactions, epithelial cells [1], cell signalling pathways [2], and ant foraging trails [3], a common formal development architecture would provide a platform for formally defined models. This is essential for verification and validation, important for future models having a part in drug development, and for advancing collaboration and understanding between researchers. By describing agents as autonomous communicating machines, agent models are inherently parallel, an essential feature when trying to run future simulations of millions of cells in a tissue model. A framework is in development for this agent representation. This includes an agent representation specification, a parser that translates agent descriptions into executable code for serial and parallel computers, and results analysers and viewers. The visual results of a model can be seen in Figure 1 for the intra-cellular NF-κB pathway.

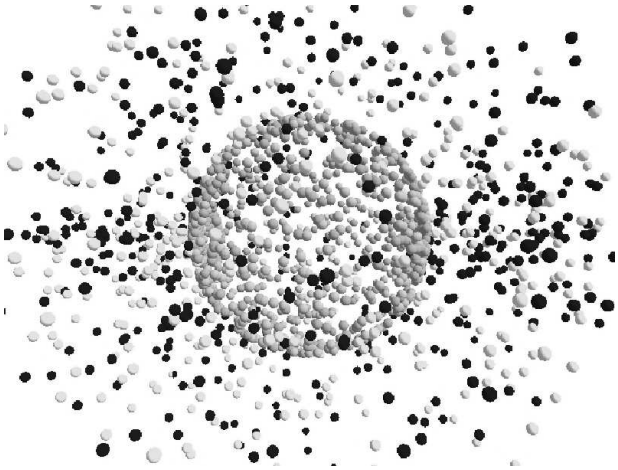

Figure 1. Visualisation of agents in the NF-κB pathway model showing proteins in the cytoplasm and the nuclear receptors on the surface of the nucleus [2]

The main reasons to create a formal agent-based architecture include:

- the ability to understand biological systems not just as components but as a system
- the provision of a clearer mode for collaboration between modellers and biologists by the use of a one-to-one mapping of biological entities to computational agents
- the creation an open format for better collaboration and understanding
- the use of formally defined agents as a basis for the validation and verification of models

Agent-based models are also intrinsically parallel which supports the aim of building more comprehensive models with large numbers of advanced agents.

 ISBN 1-56555-303-9

The modelling architecture has been designed with running models in parallel from the onset.

2. AGENTS AS FORMAL MACHINES

In general agents in agent-based modelling are components of a processing and communicating system. Agents can hold information, receive external information, process information, and act upon any results by changing internal information and sending information externally. To formally define agents a representative model is needed that includes the ability to hold information, process information, and communicate information. Classically, cellular automata have been used to describe a simple agent system made up of state machines (a model of behaviour composed of states, transitions and actions). This model is too simple to accommodate the complexities of biological systems that we are interested in modelling. Therefore a new model is proposed based around the X-machine computational model.

2.1 Cellular automata

Cellular automata [4] can be described as a framework for defining interacting components in a system. They consist of an array of cells, each of which can be in one of a finite number of possible states. Each cell, or agent, is defined in space with predefined communications to neighbour agents. Each agent has a state that is defined in the next time step as a logical operation on its neighbours states and its own state. Each agent is formally a finite state machine. Finite state machines are defined as being comprised of a set of states, a set of input symbols, an initial state, a set of accepting states, and a set of transitions where transitions are defined from a state to another state using input symbols.

Definition. A finite state machine is an 5-tuple:

$$FSM = (Q, \Sigma, T, q_0, F)$$

where:

- Q is the finite set of states
- Σ is a finite alphabet of input symbols
- T is the function from $Q \times \Sigma \rightarrow Q$ (the transition function)
- $q_0 \in Q$ is the initial state
- $F \subset Q$ is a set of final (or accepting states)

These machines are a powerful way of describing and implementing the control logic for hardware, applications, and in this context, agents. They are powerful because they follow simple rules and are easy to verify. They are also powerful because they can be used to generate programming code.

Unfortunately non-trivial systems cannot be modelled in this way due to a lack of data representation. The number of states and the number of input symbols from neighbouring agents start to explode when additional complexity is added. For a non-trivial one-dimensional cellular automata with two states per agent and two neighbours, one per side, the combination of input symbols (the current state and two neighbour states) would be $2^3 = 8$. For a simple biological cell model in two-dimensions, a cell with eight neighbour cells, and four states, representing the cell cycle (G_1, S, G_2 and M) the number of rules needed in the transition function rises to $4^9 = 262144$.

Agents are also aligned to a grid with static communication and therefore the mobility of agents and their ability to interact with different agents is lacking in this model. This is a severe restriction when modelling migration of biological cells in a tissue model, and protein molecules moving inside a cell. A more powerful state machine is needed that adds this capability.

2.2 X-machine computational model

The X-machine computational model [5] has already been proposed for this purpose [6, 7] and as a formal model for verifying swarms [8, 9]. X-machines are similar to finite state machines, however, X-machines differ in that they have the addition of memory so that transitions between states can include the memory and the modification of it.

An X-machine describing a biological cell would include memory variables holding information about the cell cycle, cell position, cell size and cell bonds. The transition between states, which can now operate on the memory, would include transitions that update the cell cycle, move the cell by updating the cell position, grow the cell by updating the cell size, and create and destroy new and old bonds between cell neighbours.

A particular type of X-machine has formed the basis for a specification and modelling language to define and validate software systems [10]. It has the ability to describe formally data types and functions in a intuitive way and is defined as follows:

Definition. A stream X-machine [10] is an 8-tuple

$$X = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$$

where:

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

From now on the term X-machine refers to a stream X-machine. This forms a basis for formally specifying agents as X-machines. The input Σ and output Γ alphabets are empty sets as the behaviour of agents is only determined by their memory values, their rules (defined in the partial (transition) functions) and the communication between agents.

2.3 Communicating X-machines

The added ability for X-machines to communicate can be achieved by using communicating X-machines. A Communicating X-machine model consists of X-machines that have the ability to exchange messages. This model can be generally defined as the tuple:

$$((C_i^x)_{i=1..n}, R)$$

where:

- C_i^x is the i-th Communicating X-machine in the system, and
- R is a communication relation between the n X-machines

There have been several attempts to formally define a communicating X-machine with different approaches to defining R. One of the most accepted approaches uses the idea of a communication matrix which acts as the means of communication between X-machines [11]. In this approach the matrix cells contain messages from one X-machine to another, i.e. cell(i,j) contains a message from X-machine$_i$ to X-machine$_j$. The approach of using an all encompassing communication matrix is not well suited to the use of X-machines as agents because:

1. For each agent added to a simulation the number of interactions between agents rises quadratically $O(n^2)$ which directly affects the size of the communication matrix.
2. The communication matrix therefore quickly becomes too large to store and handle and as agent communication is usually localised, nearly all matrix cells would be redundant (sparse).
3. There is no mechanism defined to add newly created agents or remove existing ones, that is to say adding and removing rows and columns from the matrix. This would be fundamental to agents representing biological cells as they may divide to produce two daughter agents or die therefore removing the agent.

In relation to agent modelling frameworks this is more suitably called an interaction matrix where agents have direct access to each others memory. This can be dangerous as formally defined agents can have their memory changed without a formal way for the agent to know about it. This can be seen as analogous to objects in the object-oriented programming paradigm where direct access to an object's variables is considered dangerous and public 'set' methods should be used instead to access privately declared variables. As communicating X-machines only communicate via messages this also is a way to stop direct access to agent's variables.

When communicating X-machines are used to represent agents in an agent-based model, communication is usually restricted to interactions with neighbouring agents that are located close to one another. Two examples of rules which specify when localised communication can take place are 1) the distance between two agents must be less than a specified maximum or 2) the agents must be in contact with each other.

2.4 Communicating Messages Via Lists

Newly proposed is the idea of the communication relation R as a collection of global message lists with each list representing a specific type of message that

a communicating X-machine can use for communication. Specific message types are defined by the information they can hold, for example position information or possible interaction and behaviour information. The messages that are sent to a message list can be read by all of the X-machines. This method is an input centric implementation. In contrast to the communication matrix where agents would need to know where to send a message to, the X-machine reading the messages from a list must use its own rules. An example is the communicating agent within maximum communication distance, to decide on either discarding or processing the message. By giving messages the ID of the agent the message is intended for, the same functionality is obtained as if the message was placed in a communication matrix cell.

An X-machine agent model of a biological intracellular pathway would include components such as protein molecules and nuclear importing and exporting receptors [2]. Each of these components would be modelled as a different X-machine agent. The types of messages that would be used for them to communicate would include position messages and bonding messages. Position messages would be used to determine if an agent is near enough to another agent and if a bond is possible. Bond messages would then be used to communicate this bond and agents would update their memory to acknowledge this fact.

These message lists then have the ability to be easily localised. There can be many message lists each having local agents. These local agents only need to send messages to the local message lists as most communication is between agents located near to each other. If a message is sent to a local message list that can affect an agent on a different message list, the message needs to be sent to that list to keep consistency.

In practice the model space is split up into space partitions, sometimes referred to as domains. In relation to this model each space partition has its own message lists representing each type of message that can be sent. Agents that fall into a space partition are associated with it and the corresponding message lists. When an agent sends a message it is automatically placed on the local message list, but if an agent is close to the edge of the space partition (referred to as the halo region) and has the possibility of affecting an agent on a neighbouring space partition a copy of the message is sent to that space partition to place on its local message list. This idea of only sending messages between space partitions when necessary translates well when space partitions are placed on separate nodes on a parallel computing cluster because it is the communication between nodes that can be the bottle neck in the computation time.

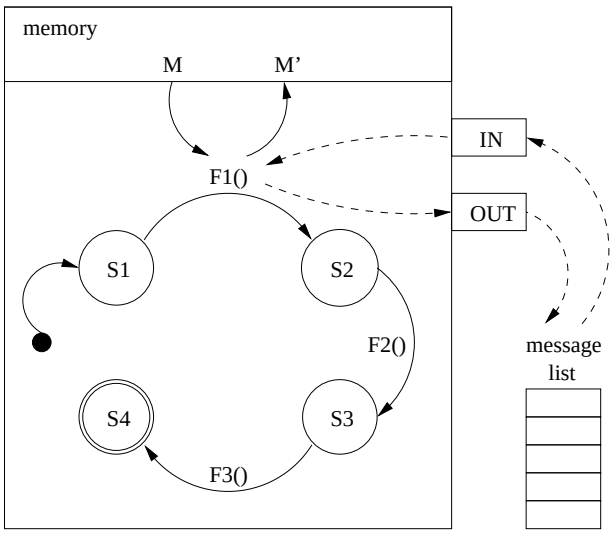

Figure 2. Abstract X-machine model

2.5 Architecture Model

Figure 2 is an abstract representation of the model. It describes the X-machine agent with its memory, system states, transition functions between the system states, and input and output ports. The figure currently shows the transition between system state $S1$ (the initial state q_0) and $S2$ via the transition function $F1$. This transition function contains rules that can then change the agents memory, M to M', and send and receive messages via the input and output ports. These in turn are connected to message lists that hold the messages used for communication between agents.

The simulation runs of a model are currently discrete time step based similar to cellular automata. But because agents are reliant on communication of messages there is a global time step after all the X-machines in a model have completed one transition function. For example the first transition function of X-machines in a model could correspond to agents sending out information (there is no restriction to a transition functions access of memory or sending and receiving messages). Each X-machine agent is processed in turn (the order is randomised each time so that no agent has priority over any other) by completing a transition function. Once all X-machines in

the model have been processed there would then be a global time step for messages to be sent to respective message lists. The second transition function could then be processed on each X-machine agent which could receive any messages and respond to them. Effectively the end and the start of transition functions act as a way to synchronise the processing of X-machines and the communication of messages.

By separating the behaviour of agents into separate functions, for example one to handle movement, modellers can interchange these functions easily with different versions with few changes needed to the rest of the model description.

With respect to cellular automata and finite state machines, the X-machine agent model provides:

- data representation in the form of memory, and
- a communication relation that is not static between agents.

This provides a formal yet very expressive way to formalise agents that can have many attributes (states), move freely (not aligned to a grid) and communicate dynamically.

3. X-MACHINE AGENT REPRESENTATION

Now that an agent model has been established an agent representation is needed so that models are easy to share and work with. Extensible Mark-up Language (XML) is capable of describing many different types of data. It provides a standard way to share structured text. By having a defined way to structure an agent, agent-based models can be created, edited, shared, and merged between modellers. By describing an agent as a communicating X-machine the model can take advantage of existing communication and modelling systems already built to run X-machine agents on different platforms. By creating an open standard for the structure of an X-machine agent new tools can be built to inter-operate with the proposed standard. The current design is very straight forward and uses simple nested XML tags to describe the structure of an X-machine. Before the X-machine is defined, 'environment' variables and functions can be defined. These are values and functions that can be used by transition functions of the X-machine. They typically include static values like π and functions that are called more than once by the X-machine. Environment functions can also be used to make the code of the transition functions more readable. The X-machine is then divided into its constituent parts:

```
<xmachine>
<memory></memory>
<states></states>
<functions></functions>
</xmachine>
```

The X-machine is defined between `<xmachine>` and `</xmachine>`. The X-machine's memory variables can then be defined between the memory tags. As the X-machine will eventually be run as part of a simulation the variables have to be well defined. At the moment this is done with respect to C variables. Variables are defined as having a fundamental type and a name. For a simple biological cell model the following memory would allow the representation of a cell as a point in space with a certain radius and in a certain point in the cell cycle. The inclusion of an ID variable allows tracking of each individual cell.

```
<memory>
<var><type>int</type><name>id</name></var>
<var><type>int</type><name>cell_cycle</name></var>
<var><type>double</type><name>x</name></var>
<var><type>double</type><name>y</name></var>
<var><type>double</type><name>radius</name></var>
</memory>
```

This example shows variables in the X-machine's memory that are of type integer or double and can be referenced by their name, for example 'ID'. Other variables can be added in the same way. At the moment only fundamental C types are handled but there is scope to add other data-types, such as arrays.

X-machine states are described inside state tags. States have fields describing their name, attributes like the initial state, and any transition functions to other states. The following state describes the 'send_location' state which is the initial state. It has one transition function called 'send_location_message' with destination 'read_locations' state.

```
<state>
<name>send_location</name>
<attribute>initial</attribute>
<trans>
<func>send_location_message</func>
<dest>read_locations</dest>
</trans>
</state>
```

Any transition functions mentioned in the X-machine states have to be defined as an X-machine function.

The function names have to relate to the names used already. For example the 'send_location_message' can be defined thus:

```
<function>
<name>send_location_message</name>
<code><![CDATA[
   add_location_message( get_id(),
                         get_cell_cycle(),
                          get_x(),
                           get_y(),
                            get_radius() );
]]></code>
</function>
```

As part of a biological cell model this would be the first function and every cell (agent) would send a message to the message list that contains their ID, position, size, and location in the cell cycle. In a second function each cell would then read the message list to determine the cells in its local neighbourhood and for example if it is overlapping with any cells or if any bonds are possible. Further functions could handle the creation of bonds (possible additional memory variables would be needed for cells to register these, and an additional message type 'bond' for agents to communicate this information) and the movement of cells.

The code tagged field used in specifying functions is used to hold the C code describing what rules to follow. The code field adheres to C functionality so comments can be added to it. Code fields are surrounded by 'CDATA' tags so that any XML parser does not parse C code as XML. The above code describes creating a new location message, assigning its variables to hold information from the current agent, and sending the message. This and other inbuilt functions handle processes that are a part of the architecture like sending messages, receiving messages, creating new agents, removing agents, and accessing memory values.

After every X-machine transition function is accounted for the X-machine is defined. Lastly the messages that can be sent and received need to be well defined also. Each message is defined inside a message tag, is given a name, and any variables it needs to hold. The message defined below refers to the message used in the above X-machine function.

```
<message>
<name>location</name>
<var><type>int</type><name>id</name></var>
<var><type>int</type><name>cell_cycle</name></var>
<var><type>double</type><name>x</name></var>
<var><type>double</type><name>y</name></var>
<var><type>double</type><name>radius</name></var>
</message>
```

The format for the message variables is the same format as the variables defined in the X-machine's internal memory. Many types of messages can be defined with variables for different purposes.

4. X-MACHINE AGENT IMPLEMENTATION

Because the agent-based model is now formally defined, tools can be created to inter-operate with it. Models can be edited by directly changing the XML description file with a text editor or with a graphical user interface. A windows based editor is more accessible and can be created to hide any detail a modeller does not need to see. Also editors can be created for certain types of models, for example cellular protein molecules or tissue cells, so that previously defined specific functions are already there for the modeller to use. At the moment an editor is in production to edit all aspects of the model XML description file. A parser has been created that converts the model description (in XML) to a runnable C program via a C compiler. The parser can produce two versions, a serial version and a parallel version that uses the message passing interface (MPI) library to send messages between nodes on a computing cluster.

When a model is to be created and solved using the currently implemented architecture, the X-machine agent definitions are first loaded from a predefined XML formatted text file. This description file specifies the memory set of the X-machine 'M', the set of processing states the X-machine can be in 'Q', the initial state q_0, the next state partial function 'F', and the communication relation which is taken to be the types of messages that can be sent 'R'. As it is only the behaviour of agents that are loaded from the XML file, an additional start-up file is required to specify each agent's initial condition before a run of the model can begin. This start-up file is also an XML formatted text file and defines the initial memory state of agents 'm_0'. A specially developed parser takes an X-machine agent description and produces an executable program (compiled C code) that can accept a start-up file, run the X-machine agents, and implements the global message list communication relation. The parser simply takes the information about the X-machine agents in a model and any messages that can be sent and applies these to a program template. In theory the program can be written

in other programming languages but C is used at the moment.

The model will run for the prescribed number of iterations given at start time. At each iteration of the model run, an XML file is saved that contains the new memory state of the agents in the model. Each of these files can then be used as the start of a new run of the model or for the extraction and analysis of data. The extracted data can be analysed and displayed as graphs or used in visual representations of the agents. Tools have been created for exporting the results to external applications, allowing the results to be viewed in an interactive three-dimensional animation program. Although viewing the agents in two and three dimensional animations does not give concrete results, it can be invaluable for the modeller and biologists to understand what is happening in a simulation and to communicate ideas. Images and videos from models in the application section can be viewed at http://www.dcs.shef.ac.uk/stc/x-agents/

The parallel version of the X-machine agent architecture has been developed on a dual processor (with hyper-threading to make it act like a quad processor) desktop machine. The proposed parallel architecture has been designed to be very scalable. There is current access to a 128 processor cluster with plans to run models in the near future and talks have already started with HPCx (http://www.hpcx.ac.uk/) to run models on part of their 1536 processor cluster.

5. X-MACHINE AGENT APPLICATIONS

The architecture described has been used to model the social behaviour of epithelial cells, the NF-κB signalling pathway, and the electrical charge of cardiac cells.

The Epitheliome Project at the University of Sheffield, UK, and University of York, UK, aims to develop a computational model that is able to predict the social behaviour of cells in epithelial tissues [1, 12]. The X-machine agent architecture is being used to help develop this model. Each epithelial cell is represented by an X-machine agent with memory holding information, for example the cell position, cell size, number of bonds, and position in cell cycle. Messages are used to communicate size and position and the making and breaking of bonds.

A model of the NF-κB pathway (a cell signalling pathway that is vital to immune response regulation) is being developed [2] and has been partially transfered onto the X-machine agent architecture. Each protein molecule and nuclear receptor is represented by an X-machine agent with memory holding information about position, velocity, bound state, and bonding interaction radius. Messages are used to communicate position and possible bonds. Figure 1 shows a three-dimensional visualisation of the results created using a purpose built viewing tool.

Another project has used X-machine agents to define a layer of cardiac cells in order to simulate the electrical action potential that is diffused through them during normal and abnormal heart function [13]. Messages in this model are used to communicate electrical charge in each cardiac cell. Although the present model is grid aligned and similar to a cellular automata simulation there is scope to arrange the cardiac cells in the simulation into the shape of a heart by adjusting the starting positions in memory of the agents in the initial agent states file.

6. CONCLUSIONS AND FUTURE WORK

In this paper the practicalities of using communicating X-machines to define agents in agent-based modelling are described. A specific area of interest is the different ways messages can be used as a communication relation between agents and how this is ideally suited to creating an architecture that can take advantage of parallel computing machines and systems. For example instead of a message list structure, a tree structure could hold information about the relation between agents as well as the messages between them. And the idea of a node holding lists of X-machine memory, system states, functions, and passing messages between nodes using MPI leads to the formalisation of nodes as communicating X-machines in themselves.

The currently implemented architecture is explained with several models mentioned as examples of its practical application but more in-depth case studies are needed to fully document the use of the architecture and the use of formal methods. Future work of the model and its implementation will involve:

- fully documented case studies of agent models and their simulation results,
- the experimentation of different communication relations between communicating X-machines,
- the use of X-machine agents as a formalism with analytic results used to help verify or guide the simulation efforts,

- scaling up the parallel version for running simulations with hundreds of thousands of agents,
- and the creation of tools for modellers to use, create and edit X-machine agent models.

ACKNOWLEDGEMENT

The authors would like to thank those who provided the models for testing of the architecture, Dawn Wood for the epitheliome model, Mark Pogson for the NF-κB pathway model, Susheel Varma for the cardiac cell model, and Phil McMinn for the feedback on the framework. Also those on the Epitheliome Project and in the Computational Systems Biology research group for providing valuable input. Simon Coakley is funded by an EPSRC studentship. The Epitheliome Project is funded by EPSRC, and NF-κB pathway research is funded by BBSRC and MRC.

References

[1] DC Walker, G Hill, SM Wood, RH Smallwood, and J Southgate. Agent-based computational modeling of wounded epithelial cell monolayers. *IEEE Transactions in NanoBioscience*, 3(3):153–163, September 2004.

[2] M Pogson, E Qwarnstrom, R Smallwood, and M Holcombe. Formal agent-based modelling of intracellular chemical reactions. *Biosystems (to appear)*, 2006.

[3] DE Jackson, M Holcombe, and Ratnieks FLW. Trail geometry gives polarity to ant foraging networks. *Nature*, 432:907–909, December 2004.

[4] S Wolfram. *A New Kind Of Science*. Wolfram Media, 2002.

[5] S Eilenberg. *Automata, Languages, and Machines*. Academic Press, 1974.

[6] M Holcombe. X-machines as a basis for dynamic system specification. *Software Engineering Journal*, 3:69–76, 1988.

[7] E Kehris, G Eleftherakis, and P Kefalas. Using X-machines to model and test discrete event simulation programs. *Proceedings of 4th World MultiConference on Circuits, Systems, Communications & Computers*, 2000.

[8] CA Rouff, MG Hinchey, W Truszkowski, and JL Rash. Verifying large numbers of cooperating adaptive agents. *11th International Conference on Parallel and Distriuted Systems*, June 2005.

[9] C Rouff, W Truszkowski, J Rash, and M Hinchey. Formal approaches to intelligent swarms. *28th Annual NASA Goddard Software Engineering Workshop*, 2003.

[10] M Holcombe and F Ipate. *Correct Systems: Building A Business Process Solution*. Springer-Verlag, 1998.

[11] T Balanescu, AJ Cowling, M Georgescu, M Holcombe, and C Vertan. Communicating stream X-machines are no more than X-machines. *Journal of Universal Computer Science*, 5(9):494–507, September 1999.

[12] DC Walker, J Southgate, M Holcombe, DR Hose, SM Wood, S Mac Neil, and RH Smallwood. The epitheliome: Agent-based modelling of the social behaviour of cells. *Biosystems*, 76((1-3)):89–100, August 2004.

[13] RH Clayton. Computational models of normal and abnormal action potential propagation in cardiac tissue: linking experimental and clinical cardiology. *Physiological Measurement*, 22(3):R15–R34, 2001.