A General Framework for Agent-based Modelling of Complex Systems

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

Abstract

Agent-based approaches to modelling complex systems and phenomena are becoming popular and are proving successful in a number of areas. However, the underlying basis of these techniques is sometime rather 'ad-hoc' and the models are often only applied to specific systems. This paper describes a *general* approach that is based on the use of fully general computational models, using a formal model of an agent and a rigorous approach to building systems of communicating agents within virtual environments, a technology which is, nonetheless, accessible to researchers in experimental biology and medicine. A collection of tools has been built which allow for efficient simulation of such systems and their visualisation. Included in this work is the implementation of the simulations on parallel clusters of computers to enable large numbers of agents to be simulated. Application areas where the method has been successfully applied include biology, medicine and economics. In the former the detailed models have led researchers to fundamentally new discoveries.

Keywords: X-machines, formal models, agent simulation architectures

1 INTRODUCTION

The development of agent-based systems has been a major interest to researchers in a number of application fields both within Computer Science and in other subjects. Often, the systems are built in an *ad hoc* manner and are very difficult to maintain and extend by people who are not directly involved in their construction. For the simulation of complex systems, however, only a few approaches have really generated significant results that really deliver new insights, for example in biology and economics. The framework described here has been used, successfully, by biologists to uncover new aspects of biology and a better understanding of the processes underlying some biological systems, Walker et al (2004a, 2004b), Pogson et al (2006), Qwarnstromet al (2006), Jackson et al.(2004).

There are many agent frameworks proposed for various application domains, but few share the flexibility or the power of the approach proposed here. In this respect, Kefalas et al. (2003) describe a formal basis for the development of an agent-based simulation framework based round the concept of a communicating X-machine. This idea has now been taken further and a prototype framework has been built and is being evaluated in a number of case studies. Work has also been done on converting the software to run on a parallel computing environment.

2 REQUIREMENTS FOR A FRAMEWORK

In order to develop an extensible framework which allows for the construction of simulation environment that will enable the creation and operation of millions of autonomous agents it is necessary to take a fundamental look at the theoretical underpinnings of the issue and to choose as a design metaphor concepts that will allow for highly efficient simulations and with the potential to exploit parallel computer systems.
The following set of desirable features reflect not only the practical issues facing agent-based modelling but also the principle scientific needs of providing suitable information to enable other researchers to investigate the mechanics of the model, to adapt and to extend it and, in short, to *trust* that it is founded on a coherent, consistent and realistic set of principles.

1. Each agent needs to be defined in a way that is as general as possible and in a language that is abstract and mathematically founded. A specification language is needed rather than using working source code or pseudo code or outline algorithms.
2. The environment of the model also needs to be defined precisely - this includes the space within which the agents act, the nature of the communication that exists between the agents and the external environmental that will influence the model.

3. A standard framework for compiling the agent specifications onto a suitable platform for computation. This is likely to require mechanisms for utilising parallel and Grid computing in order to deal with realistic numbers of agent - millions rather than thousands.
4. In an ideal world a strong framework would lead to an exchange mechanism whereby researchers would make their agent descriptions available to others to use within their simulations.
5. The mathematical analysis of complex systems is extremely difficult and the use of formally defined agents could be a basis for the validation and verification of models in the longer term.

In this paper we address these issues and provide a framework, FLAME (FLexible Agent-based Modelling Environment), a specification language, XMML and tools to compile the specified agent-based systems into code optimised for efficient parallel processing. The framework described here allows modellers to define their agent based systems and automatically generates C code of a highly efficient nature. This has been ported onto a parallel computing cluster using MPI (Message Passing Interface). The systems allow for the detailed validation, systematic and formalised simulation and testing based on previous work by Holcombe (1998), Kefalas et al (2003a, b), Eleftherakis et al (2003). Details of FLAME will be found at www.FLAME.ac.uk

The language of X-agents, XMML, described here provides for a precise definition of both the individual and collective behaviour of agents by specifying their state transitions, internal memory and communication protocols. An important issue that we have had to face is that agents in biological simulations have varied life styles, so, for example, cells divide, new agents appear and disappear so the situation is very fluid. This poses some important issues in terms of defining standards for the field. Many of the existing standards, such as BML, CellML etc. focus on the definition of data, which is static in its structure, rather than also looking at process definition. This is something that XMML attempts to deal with. Using the XMML language we can build converters and translators to any other framework and tool, this is because XMML is based on a fully general – Turing computable – computational model.

Modern developments in biology, economics etc. mean that we have to tackle the issue of *extremely* large amounts of data throughput which is done by providing efficient data transfer mechanisms and filters that permit the operation of simulation environments featuring many thousands, and shortly, millions of agents, Walker et al, (2004a, b). The potential for using agent-based modelling in biology is large and the diversity of different application areas is considerable. Examples given here cover the modelling of cellular interactions, epithelial cells Walker at al (2004), cell signalling pathways Pogson et al (2006) and ant foraging trails Jackson et al (2004). A common formal development architecture is essential and would provide a platform for formally defined and analysable models. This is essential for satisfactory verification and validation of the models, is important for future applications where models may play a part in drug development, and for advancing collaboration and understanding between researchers.

3 AGENTS AS FORMAL MACHINES

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, and 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 capabilities can be seen in examples later. Agent-based systems are also intrinsically parallel which supports the aim of building more comprehensive models with large numbers of advanced agents. The modelling architecture has been designed with the running of models in parallel on parallel computers from the onset. Agents in agent-based modelling are components of a parallel processing and communicating system. To formally define agents a model is needed to represent them that includes the ability to process information and communicate. 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 approach is proposed based around the X-machine computational model.

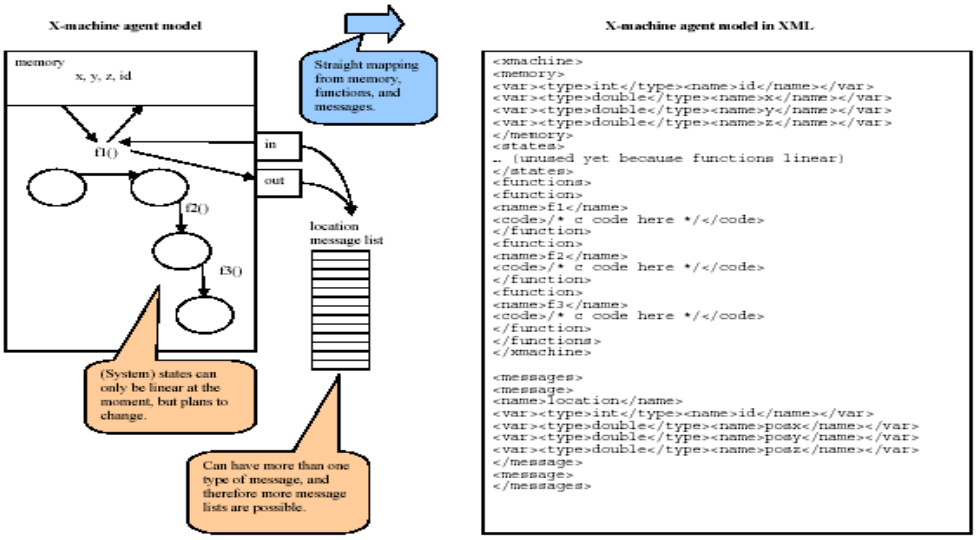
X-machine computational model

The X-machine computational model Eilenberg (1974) has already been proposed for this purpose Holcombe (1988), Holcombe and Ipate (1998) and as a formal model for designing and verifying swarm satellite systems at NASA, Hinchey et al (2005). 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.

A stream X-machine is a generalised finite state machine which has the added element of an internal memory that influences the operation of the transitions and thus the behaviour of the machine. This type of machine is extremely general and has been studied in some detail. The added ability for X-machines to communicate can be achieved by using communicating X-machines Balanescu et al (1999). However, 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. We propose 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, for 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 cell's intra-cellular pathways would include components such as protein molecules and nuclear importing and exporting receptors. 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 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 in the sense that the location of the agent is utilised when the messages are being interpreted on a parallel machine where agents inhabit different processors. 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. As long as messages from an agent on one message list that can affect an agent on a different message list are sent to that message list there is 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 there is a necessary need 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 1 is an abstract representation of the model. It describes an 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 (the initial state) and via the transition function. This transition function contains rules that can then change the agents memory, to, 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.

4 X-MACHINE AGENT REPRESENTATION DETAILS: XMML

Now that an agent model has been established an agent representation is needed so that models are easy to share and work with. The 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 having an open standard for the structure of an X-machine agent new tools can be built to inter-operate with the standard. The current design of the standard 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 and functions that are called more then 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 its tags `<xmachine>` and `</xmachine>`. The first part of the X-machine that is defined is its memory. 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 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.

5 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 the modeller does not need to see. Also, editors can be created for certain types of models so that built-in functions are already there 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 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, 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. 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 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 and the HPCx (http://www.hpcx.ac.uk/). The new EU project EURACE will develop large scale models of economic systems using the FLAME framework.

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 demonstrated with several models shown as examples of its practical application but more in-depth case studies are needed to fully document the use of the architecture. 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,
- 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.

Bibliography

Balanescu et al (1999) 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.

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

Eleftherakis et al (2003). G. Eleftherakis, P. Kefalas "An agile formal development methodology", in Tigka K. and Kefalas P. (Eds.), *Proceedings of the First South-East European Workshop on Formal Methods, Thessaloniki, Greece, 20 November 2003, Thessaloniki*, pp. 119-137.

Hinchey et al (2005) M Hinchey, C Rouff, W Truszkowski, J Rash Requirements of an Integrated Formal Method for Intelligent Swarms FMICS'05, September 5–6, 2005, Lisbon, Portugal.

Jackson et al (2004) DE Jackson, M Holcombe, and Ratnieks FLW. Trail geometry gives polarity to ant foraging networks. *Nature*, 432:907-909, December 2004.

Kefalas et al (2003a) Kefalas P., Holcombe M., Eleftherakis G., Gheorghe M. (2003), "A formal method for the development of agent-based systems", in *Intelligent Agent Software Engineering* (V. Plekhanova Ed.), Idea Group Publishing, pp. 68-98.

Pogson et al (2006) Pogson et al 2006, M. Pogson, M. Holcombe, R. Smallwood, E Qwarnstrom, Formal Agent-Based Modelling of Intracellular Chemical Interactions, B*ioSystems*, to appear.

Qwarnstrom et al (2006). E Qwarnstrom, M. Pogson, M. Holcombe, R. Smallwood. "Predictive Agent-Based NF-{kappa}B Modelling - Involvement of the Actin Cytoskeleton in Pathway Control", Submitted

Walker et al (2004a) 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.

Walker et al (2004b) 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.