How to build and use agent-based models in social science

Nigel Gilbert

Centre for Research on Simulation for the Social Sciences,

School of Human Sciences,

University of Surrey,

Guildford GU2 5XH,

UK

n.gilbert@soc.surrey.ac.uk

Pietro Terna

Dipartimento di Scienze economiche e finanziarie,

Università di Torino,

corso Unione Sovietica 218bis,

10134 Torino,

Italy

pietro.terna@torino.alpcom.it

May 18, 1999

Abstract

The use of computer simulation for building theoretical models in social science is introduced. It is proposed that agent-based models have potential as a 'third way' of carrying out social science, in addition to argumentation and formalisation. With computer simulations, in contrast to other methods, it is possible to formalise complex theories about processes, carry out experiments and observe the occurrence of emergence. Some suggestions are offered about techniques for building agent-based models and for debugging them. A scheme for structuring a simulation program into agents, the environment and other parts for modifying and observing the agents is described. The article concludes with some references to modelling tools helpful for building computer simulations.

Keywords: agent based computational economics, social simulation, neural networks, classifier systems, genetic algorithms.

1 Computational modelling as a third way of building social science models

1.1 Building models

Almost all social science research proceeds by building simplified representations of social phenomena. Sometimes these representations are purely verbal. For example, the traditional work of historical scholarship is a book-length representation of past events, abstracted and simplified to emphasise some events and some inter-relationships at the expense of others. The difficulty with such verbal presentations is that it is hard for the researcher and the reader to determine precisely the implications of the ideas being put forward. Are there inconsistencies between the various concepts and relationships? Can they be generalised and if so, what inferences can one make?

In other fields, for example, some areas of economics, the representation is usually much more formal and often expressed in terms of statistical or mathematical equations. These make assessing consistency and generalisability and other desirable properties much easier than with verbal representations. In these areas, it is generally accepted that understanding the social world involves model-building. However, statistical and mathematical models also have some disadvantages. Prime among these is that many of the equations which one would like to use to represent real social phenomena are simply too complicated to be analytically tractable. This is particularly likely when the phenomena being modelled involve non-linear relationships, and yet these are pervasive in the social world. The advantages of mathematical formalisation thus evaporate. A common solution is to make simplifying assumptions until the equations do become solvable. Unfortunately, these assumptions are often implausible and the resulting theories can be seriously misleading. This is the criticism often put before economists who make assumptions such as the availability of perfect information, perfect rationality and so on, not because they believe the economic world really does have these characteristics, but because otherwise their models cannot be analysed.

Ostrom (1988) proposed that there are three different 'symbol systems' available to social scientists: the familiar verbal argumentation and mathematics, but also a third way, computer simulation. Computer simulation, or computational modelling, involves representing a model as a computer program. Computer programs can be used to model either quantitative theories or qualitative ones. They are particularly good at modelling processes and although non-linear relationships can generate some methodological problems, there is no difficulty in representing them within a computer program.

The logic of developing models using computer simulation is not very different from the logic used for the more familiar statistical models. In either case, there is some phenomenon that we as researchers want to understand better. This is the 'target'. We build a model of the target through a theoretically motivated process of abstraction (this model may be a set of mathematical equations, a statistical equation, such as a regression equation, or a computer program). We then examine the behaviour of the model and compare it with observations of the social world. If the output from the model and the data collected from the social world are sufficiently similar, we use this as evidence in favour of the validity of the model (or use a lack of similarity as evidence for disconfirmation).

If the model is a mathematical equation, it may be possible to infer its behaviour by a process of mathematical reasoning. If the model is a statistical equation, we run it through a statistical analysis program such as SPSS. If the model is a computer program, its behaviour can be assessed by 'running' the program many times to evaluate the effect of different input parameters. It is the behaviour of the program that we call a computer simulation. In all these cases, we need data about the target. For example, we might be interested in spread of opinions within a community of people. We would conduct surveys, perhaps several over a span of time, to measure the community members' opinions (the dependent or output variable) and a number of variables thought to influence their opinions (the independent or input variables). The model would then be 'run'. If the model consists of a regression equation, this amounts to deriving a vector of expected values of the dependent variable, given some measured values of the independent variables. If the model consists of a computer program, it consists of running the program with a variety of input parameters and observing the program's outputs (Bratley, Fox and Schrage 1987). Finally, the values of the output variable derived from the run are compared with corresponding values of the independent variable measured from the target. (See Gilbert and Troitzsch 1999 for an extended discussion).

1.2 The uses of simulation

Computer simulation has been widely adopted in the natural sciences and engineering as a methodology (Zeigler 1976), but is a comparatively recent approach in the social sciences. This may be because the principal value of simulation in the social sciences is for theory development rather than for prediction. In engineering, the usual purpose of a simulation of, for example, the trajectory of a space station is to predict its future position. Similar types of prediction are unusual in the social sciences. Examples where simulation has been used for prediction are in demography, where the aim is to predict the age and sex structures of a population in the future, and for 'microsimulation' to predict, for example, future dependency ratios as the proportion of the population at work decreases and the number of elderly people increases.

Recently, however, the use of simulation in the social sciences has undergone major growth for two linked reasons. First, there has been the development of agent-based simulations, considered in more detail in the next section. Second, the value of computer programs as models for discovery, understanding and formalisation has been better appreciated. For example, it has been discovered through simulation that given certain kinds of relationship between people's purchasing decisions, under some wide range of conditions, after a short time people will prefer one of two competing products and that this preference will persist (a phenomenon called "lock-in" by Arthur, 1989). A well-known instance is the competition between the VHS and Betamax video tape formats.

Another example of the use of simulation is the discovery that, if one person's attitude on some issue (e.g. Left vs. Right or a preference for beer rather than wine) influences others' according to a non-linear relationship (we do not need to specify very closely what this relationship is, other than that it is not linear), we can expect clusters of people sharing the same attitude, but also the persistence of a minority with the other preference (Nowak and Latané, 1994). In both these examples, it is not possible to use the models to make predictions (the model does not tell us which tape format or whether wine and beer will be favoured), but the models do point to classes of those observations (e.g. lock-ins) which could be expected (and rule out others) and illustrate a mechanism through which those observations could arise.

Once one has a model, the fact that it is formulated as a computer program will, in principle at least, allow it to be communicated to others. In practice, it is rare for programs themselves to be published in the literature. More commonly, a high level rendering of the algorithm, perhaps in 'pseudo-code', is made available. There are several reasons for this. Perhaps the least defensible is programmers' reluctance to expose their coding style to the critical gaze of others. Once a program has been experimented with, debugged and used to generate results for a range of inputs, the code is often not in the elegant shape that the author would have liked. A better reason for publishing only a high level version of the code is that even today, programs are often not easily portable from one machine and one operating system to another. A program written to work on a PC in C++ will probably not work first time when run on a Unix machine. There is a solution to this now emerging: the use of special toolkits and libraries (e.g. Swarm, http://www.santafe.edu/projects/swarm, and SDML, Moss et al 1998) that both provide a suite of functions useful for the programming of social simulations and a guarantee of machine independence.

1.3 From top to bottom and back again

The breakthrough in computational modelling in the social sciences came with the development of multi-agent systems (MAS). These models were derived from work in a sub-area of artificial intelligence called distributed artificial intelligence (DAI). DAI aimed to solve problems by dividing them amongst a number of programs or agents, each with its own particular type of knowledge or expertise. In combination, the collection of agents would be better at finding solutions than any one agent working on its own. While DAI is primarily concerned with engineering effective solutions to real world problems, it was soon noticed that the technology of interacting intelligent agents could be applied to modelling social phenomena, with each agent representing one individual or organisational actor.

The standard MAS today consists of a number of 'agents' communicating via messages passed between them through an 'environment'. All the agents and the environment are represented within one computer program. Often the environment is modelled as a two dimensional space and each agent is positioned in a different location. In some models, the agents are free to travel thorough the space while in others they are fixed. The design of the latter models is often influenced by the tradition of work on cellular automata (Hegselmann, 1996).

There is a certain amount of confusion over the term 'agent' because the basic idea has recently been applied in many different domains in addition to social science modelling. For example, agents have been built that search the Internet on the user's behalf for cheap deals. Other agents are employed as 'wizards' helping the user learn to use application programs such as word processors. From the point of view of MAS, agents are processes implemented on a computer that have autonomy (they control their own actions); social ability (they interact with other agents through some kind of 'language'); reactivity (they can perceive their environment and respond to it); and pro-activity (they are able to undertake goal-directed actions) (Wooldridge and Jennings 1995).

Because agents are computational processes intended to model, in a much-simplified way, the capabilities of humans, it is easy to drop into using a vocabulary normally reserved for describing people, such as 'intelligent', ‘intention’ and 'action'. This only becomes a source of confusion if it is then thought that there is no significant difference between, for instance, the intelligence of a computer agent and the intelligence of a person. Even the most sophisticated agent programs are probably less intelligent than an ant (although it is difficult to formulate an appropriate indicator of intelligence at this level) and ascription of intentionality to a program must always be considered as metaphorical. Nevertheless, as we shall see, there are lessons that can be learned even from such apparently crude representations of people.

The task of the modeller is thus to define the cognitive and sensory capabilities of the agents, the actions they can carry out and the characteristics of the environment in which they are located, to set up the initial configuration of the system, and then to observe what happens when the simulation is run. Generally, one is looking for emergent phenomena, that is some patterns arising at the level of the system as a whole that are not self-evident from consideration of the capabilities of the individual agents. For example, the phenomenon of 'lock-in' mentioned above is one such emergent feature. If the agents are constructed to be deliberative consumers, purchasing one brand of video tape or another because of the availability of machines on which to play it, it is not obvious from just examining individual agents that one format will win out almost completely over the other. A more formal definition of emergence is that a phenomenon is emergent if it requires new categories to describe it which are not required to describe the behaviour of the underlying components (in this case, agents). The idea of emergence has had a powerful influence on some branches of the natural sciences and it also has obvious resonances in the social sciences, where the relationship between individual action and emergent social structure is a fundamental issue. For example, institutions such as the legal system and government that are recognisable at the 'macro-level' emerge from the 'micro-level' actions of individual members of society.

In comparison with the natural sciences, a complication needs to be borne in mind when considering emergent phenomena in the social sciences. In the physical world, macro-level phenomena, built up from the behaviour of micro-level components, generally themselves affect the components. For example, the macro-level behaviour of an avalanche is constituted by the micro-level behaviour of millions of snow particles, but itself pulls more snow along with it. The same is true in the social world, where an institution such as government self-evidently affects the lives of individuals. The complication in the social world is that individuals can recognise, reason about and react to the institutions that their actions have created. Understanding this feature of human society, variously known as second-order emergence (Gilbert 1995), reflexivity (Woolgar 1988) and the double hermeneutic, is an area where computational modelling shows promise.

2 Building multi-agent systems

There is no one recognised best way of building agents for a multi-agent system. Different architectures (that is, designs) have merits depending on the purpose of the simulation. Nevertheless, every agent design has to include mechanisms for receiving input from the environment, for storing a history of previous inputs and actions, for devising what to do next, for carrying out actions and for distributing outputs. Agent architectures can be divided into those that are rooted in the symbolic paradigm of AI and non-symbolic approaches such as those based on neural nets. In addition there are a few hybrid MAS (e.g. Klüver, 1998).

2.1 Techniques for constructing agents

2.1.1 Production systems

One of the simplest but nevertheless effective designs for an agent is to use a production system. A production system has three components: a set of rules, a working memory and a rule interpreter. The rules each consist of two parts: a condition, which specifies when the rule is to executed ('fire'), and an action part, which determines what is to be the consequence of the rule firing. For example, an agent might be designed to roam over a simulated landscape, collecting any food that it encounters on its travels. Such an agent might include a rule that states (in an appropriate symbolic programming language): IF I am next to some food, THEN pick it up. This would be one of many rules, each with a different condition. Some of the rules would include actions that inserted facts into the working memory (e.g. I am holding some food) and other rules would have condition parts that tested the state of the working memory (e.g. IF I am holding food THEN eat it). The rule interpreter considers each rule in turn, fires those for which the condition part is true, performs the indicated actions for the rules that have fired, and repeats this cycle indefinitely. Different rules may fire on each cycle either because the immediate environment has changed or because one rule has modified the working memory in such a way that a new rule begins to fire.

Using a production system, it is relatively easy to build reactive agents that respond to each stimulus from the environment with some action. It is also possible, but more complicated, to build agents with some capacity to reflect on their decisions and thus begin to model cognition (e.g. Verhagen, 1999). Another possibility is to enable the agents to change their own rules using an adaptive algorithm which rewards rules that generate relatively effective actions and penalises others. This is the basis of classifier systems (for an example, see Ballot et al, 1999; the algorithm is described in Holland and Miller 1991).

2.1.2 Learning

Production system based agents have the potential to learn about their environment and about other agents through adding to the knowledge held in their working memories. The agents' rules themselves, however, always remain unchanged. For some problems, it is desirable to create agents that are capable of more fundamental learning: where the internal structure and processing of the agents adapt to changing circumstances. There are two techniques commonly used for this: neural networks and evolutionary algorithms such as the genetic algorithm.

Neural networks are inspired by analogy to nerve connections in the brain. A neural network consists of three or more layers of neurons, with each neuron connected to all other neurons in the adjacent layers. The first layer accepts input from the environment, processes it and passes it on to the next layer. The signal is transmitted through the layers until it emerges at the output layer. Each neuron accepts inputs from the preceding layer, adjusts the inputs by positive or negative weights, sums them and transmits the signal onward. Using an algorithm called the back propagation of error, the network can be tuned so that each pattern of inputs gives rise to a different pattern of outputs. This is done by training the network against known examples and adjusting the weights until it generates the desired outputs given particular inputs (Garson, 1998).

In contrast to a production system, a neural network can modify its responses to stimuli in the light of its experience. A number of network topologies have been used to model agents so that they are able to learn from their actions and the responses of other agents (e.g. Hutchins and Hazlehurst, 1995; Terna, 1997).

Another way of enabling an agent to learn is to use an evolutionary algorithm. These are also based on a biological analogy, drawing on the theory of evolution by natural selection. The most common algorithm is the genetic algorithm (GA). This works with a population of individuals, each of which has some measurable degree of 'fitness', using a metric defined by the model builder. The fittest individuals are 'reproduced' by breeding them with other fit individuals to produce new offspring that share some features taken from each parent. Breeding continues through many generations, with the result that the average fitness of the population increases as the population adapts to its environment.

For both neural networks and GAs, the experimenter has to make a decision about the scale at which the model is intended to work. For example, with a GA model, it is possible to have a whole population within each agent. The GA would then be a 'black box' mechanism used to give the agent some ability to learn and adapt (see Chattoe and Gilbert, 1997 for an example). Alternatively, every individual could be an agent with the result that it would be the population of agents as a whole that would evolve. Similarly, it is possible for either an individual agent to be modelled using a neural network, or a whole society to be represented by a network, with each neuron given an interpretation as an agent (although in the latter case, it is hard to build in all the attributes of agents usually required for multi-agent modelling).

2.2 The agent's environment

Agents are almost always modelled as operating within some social environment consisting of a network of interactions with other agents. Sometimes, however, it is also useful to model them within a physical environment that imposes constraints on the agents' location. The usual assumption is then that nearby agents are more likely to interact or are better able to influence each other than those farther apart.

Models of this kind may be built using techniques drawn from work on cellular automata. A cellular automata consists of a regular array of cells, each of which is in one of a small number of states (e.g. 'on' or 'off'). A cell's state is determined by the operation of a few simple rules and depends only on the state of its neighbouring cells and its own previous state. Cellular automata have been studied intensively by mathematicians and physicists because they are useful models of some physical materials and have some interesting computational properties. The classic cellular automata have only very simple mechanisms for determining the state of their cells, but the same principles can be used with more complex machinery so that cells represent agents and the array as a whole can be used to model aspects of a society (Hegselmann, 1998).

3 Object oriented programming in the agent based models perspective

In Sections 1 and 2 we have introduced the premises, the contents and the perspectives of the agent based simulation framework. Now we have to pay attention to the “container” used to build the experiments or, in technical terms, to the computer codes allowing us to run simulations in a machine.

For the reasons outlined in Section 1.2, the characteristics of the software we use are crucially important in assuring the success of this third way of formalising models. Only if we use high quality software we are able to communicate the details of our model, allow other scholars to replicate the results, and avoid difficulties in modifying poorly written code. The best way to improve the quality of the programming is to choose an object-oriented language. This choice simplifies the translation of the problem into a set of agents and events. From a computational point of view, agents become objects and events become steps activated by loops in the program. In addition, in a fully object oriented environment, events (or time steps) can be organised as objects. The key term here is object: a piece of code containing data and rules operating on them.

3.1 Memory and synchronisation problems

There are three different degrees of completeness in the structure of software tools for agent-based programming.

At the lowest level (e.g., using plain C) we have to manage both the agent memory structures (commonly constructed from arrays) and the time steps, using "for" loops to drive events; this is feasible, but also costly (a lot of software has to be written; many "bugs" have to be discovered).

At a more sophisticated level, employing object oriented techniques (C++, Objective C, etc.), the memory management problem can be avoided, but nevertheless stepping through simulated time still needs the activation of loops.

Finally, using a high level tool such as Swarm, both the memory management and the time simulation problems can be avoided. With such high level tools, events are treated as objects, scheduling them in time-sensitive widgets (such as action-groups).

An additional advantage of using a high level tool is that it is then possible to publish simulation results in a useful way. To quote from the Swarm documentation:

> The important part (. . .) is that the published paper includes enough detail about the experimental setup and how it was run so that other labs with access to the same equipment can recreate the experiment and test the repeatability of the results. This is hardly ever done (or even possible) in the context of experiments run in computers, and the crucial process of independent verification via replication of results is almost unheard of in computer simulation. One goal of Swarm is to bring simulation writing up to a higher level of expression, writing applications with reference to a standard set of simulation tools. (Swarm, http://www.santafe.edu/projects/swarm).

4 From simple models to complex results

Following Axtell and Epstein (1994:28), the problem of coping with complexity via agent-based modelling or simulation is that the experiments would be of little interest "if we cannot understand these artificial complex systems any better that we understand the real ones." To understand - and to allow others to understand - our work, simple but robust guidelines have to be followed.

First, we have to consider the agents as objects: i.e., pieces of software capable of containing data and rules to work on the data. The rules provide the mechanisms necessary to react to messages coming from outside the object.

Second, we have to observe the individual agents' behaviour via the internal variable states of each agent-object and, at the same time, the results arising from their collective behaviour. One important feature of the software is therefore to synchronise the experiment clocks: we have to be sure that the observations made at the aggregate level and the knowledge that we are picking up about the internal states of the agents are consistent in terms of the experimental schedules.

Now let us suppose that a community of agents, acting on the basis of public (i.e. common to all agents) and private (i.e. specific to each agent) information and of simple internal local rules, shows, as a whole, an interesting, complex, or 'emergent' behaviour. There are two kinds of emergence: unforeseen and unpredictable emergence. An example of unforeseen emergence occurs when we are looking for an equilibrium state, but some sort of cyclical behaviour appears; the determinants of the cyclical behaviour are hidden in the structure of the model. Unpredictable emergence occurs when, for example, chaos appears in the data produced by an experiment. Chaos is obviously observable in true social science phenomena, but it is not easy to reverse engineer in an agent-based simulation. For an experiment exhibiting both unforeseen and unpredictable emergence, see Terna (1998a, 1998b).

Thirdly, we need to evaluate whether the simulation is a success. A mathematical statistics approach can be used in which, according to Kleijenen (1998):

> The type of statistical test actually applied depends on the availability of data on the real system [that we are simulating]: (i) no data, (ii) only output data, and (iii) both input and output data. In case (i), the system analyst can still experiment

> with the simulation model to obtain simulated data. Those experiments should be guided by the statistical theory on design of experiments (DOE); an inferior - but popular - approach is to change only one factor at a time. In case (ii), real and simulated output data may be compared through the well-known Student t statistic. In case (iii), trace-driven simulation becomes possible.

This type of approach is potentially relevant, but often difficult to apply to an agent based computational model, where qualitative results are expected as well as quantitative ones.

We therefore have to adopt a weaker criterion. One such is to compare the actions of the agent behaving in the simulated framework with our knowledge about the way that actual agents behave. A more severe degree of assessment is to test the aggregate effects of agents' behaviour in terms of emerging structure, groupings, spatial effects, and so on.

Following Axtell and Epstein (1994), we can summarise the levels of agent-based model performance and analysis as follows.

Level 0: the model is a caricature of reality, as established with simple graphical devices (e.g. allowing visualisation of agent motion).

Level 1: the model is in qualitative agreement with empirical macro-structure, as established by plotting the distributions of some attributes of the agent population.

Level 2: the model produces quantitative agreement with empirical macro-structure, as established through statistical estimation routines.

Level 3: the model exhibits quantitative agreement with empirical micro-structures, as determined from cross-sectional and longitudinal analysis of the agent population.

Nevertheless, even if we obtain satisfactory results at level 3, a basic question may remain unsolved: the so-called many-to-one or identification problem, i.e.

the possibility of obtaining similar results with quite different agent structures. This is, however, a general problem of scientific method, not confined to the methodology of simulation.

Finally, we can observe that simulated results about collective behaviour can be as difficult to understand as reality. The problem is analogous to the classic one of understanding links between genotype and phenotype. It may be impossible to foretell what kind of phenotype emerges from an engineered genotype; this is a topic central to the Artificial Life (AL) domain.

In that context the idea is to assume that the behaviour of single agents is simple but adaptive, in order to explore the complexity of aggregate behaviour. Unexpected results can easily arise. Many AL models demonstrate, for example, recurrent relatively short periods of great variance or turbulence, apparently similar to the explosion of life-forms in the Cambrian period. It is not easy to describe such structures in precise terms, even though such an operation would be useful in order to compare the results of different models. If the adaptability allowed to the agents consists only in learning to forecast, then the results follow directly from the behavioural rules and from the learning algorithms that have been wired into the model. All the collective emergent behaviour can therefore be understood as the result of alternative learning schemes that may be imputed to the agents.

The task is much more complicated in the context of models where agents not only learn to forecast, but also have themselves to design their own behaviour. Here we must take into account separately both the variety of emergent behaviours of the agents and the aggregate effects of their different ways of acting. It is mainly in this context that we can discover the emergence of unexpected consequences of different initial settings of the model. The core explanation, and the related lesson useful for the interpretation of reality, is the discovery both of the micro-mechanisms explaining individual behaviour and of the way by which those mechanisms, even if very simple, can generate complex consequences as a result of the agents' interaction.

At present, standard tools for the interpretation of collective behaviour in agent models are lacking, but some techniques can be suggested. For example, following Beltratti et al. (1996), one could study clusters of agents to see how their behaviours can be interpreted. Or, to extend the idea of derivatives, one might think of "social derivatives", where the reactions of the system as a whole to changes in the behaviour of some particular group are analysed, keeping constant the actions of other social groups, to see which group is most likely to affect the aggregate outcome.

4.1 Guidelines in building environment, rules and agents

The problems arising when we go from simple models to complex results suggest that a crucial role for the usefulness and the acceptability of the experiments is played by the structure of the underlying models. For this reason, we introduce here a general scheme that can be employed in building agent-based simulations. Similar schemes are advocated and implemented in Moss (1998) and Gilbert and Troitzsch (1999).

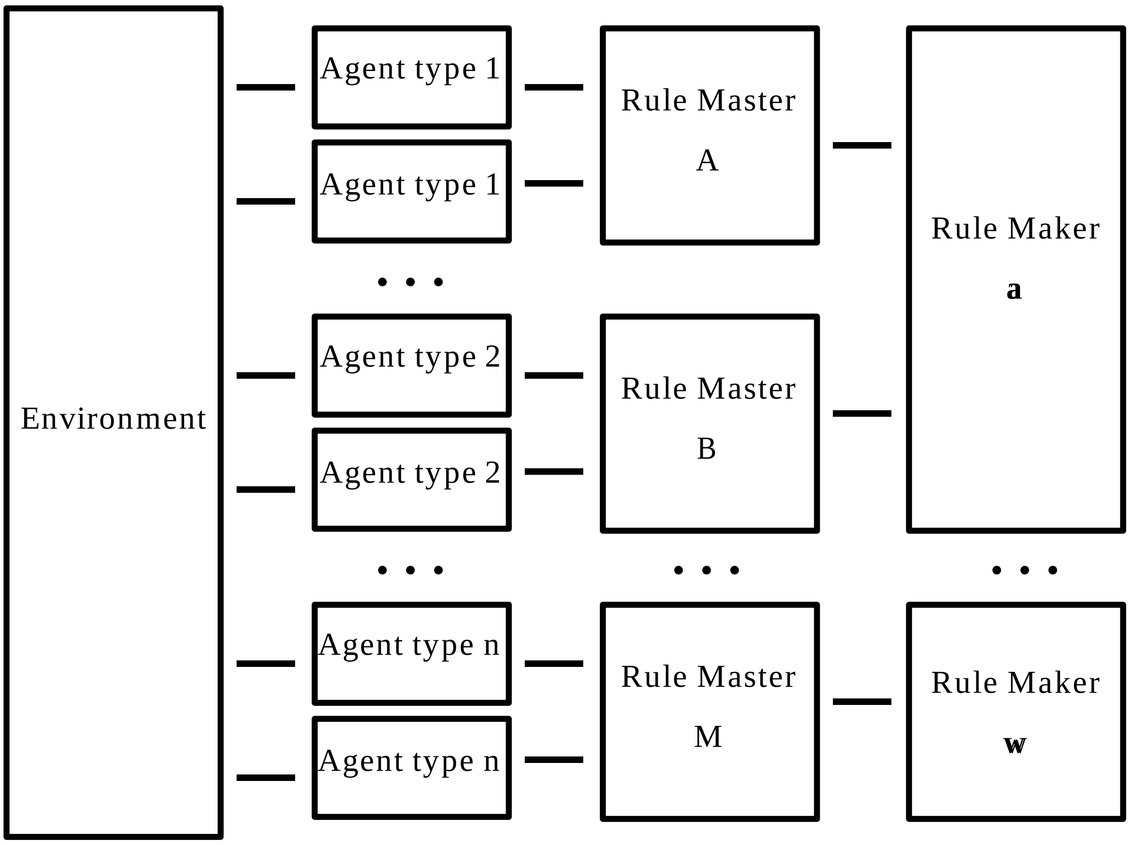

Fig.1 - The Environment-Rules-Agents framework to build agent based computational models

The main value of the Environment-Rules-Agents (ERA) scheme shown in Figure 1 is that it keeps both the environment, which models the context by means of rules and general data, and the agents, with their private data, at different conceptual levels. To simplify the code, we suggest that agents should not communicate directly, but always through the environment; as an example, the environment allows each agent to know the list of its neighbours. This is not mandatory, but if we admit direct communication between agents, the code becomes more complex.

With the aim of simplifying the code design, agent behaviour is determined by external objects, named Rule Masters, that can be interpreted as abstract representations of the cognition of the agent. Production systems, classifier systems, neural networks and genetic algorithms are all candidates for the implementation of Rule Masters.

We may also need to employ meta-rules, i.e., rules used to modify rules (for example, the training side of a neural network). The Rule Master objects are therefore linked to Rule Maker objects, whose role is to modify the rules mastering agent behaviour, for example by means of a simulated learning process. Rule Masters obtain the information necessary to apply rules from the agents themselves or from special agents charged with the task of collecting and distributing data. Similarly, Rule Makers interact with Rule Masters, which are also responsible for getting data from agents to pass to Rule Makers.

Although this code structure appears to be complex, there is a benefit when we have to modify a simulation. The rigidity of the structure then becomes a source of invaluable clarity. An example of the use of this structure can be found in the code of a Swarm application called BP-CT in which the agents are neural networks. (The code can be obtained directly from the second author or, in the future, from the 'anarchy' section of the Swarm site).

A second advantage of using the ERA structure is its modularity, which allows model builders to modify only the Rule Master and Rule Maker modules or objects whenever one wants to switch from agents based on a classifier system, to alternatives such as neural networks, production systems or genetic algorithms.

4.2 Checking the code for errors

As Axelrod (1997a:211) notes, attention to the quality of a simulation model is important throughout the research enterprise.

> In order to be able to achieve the goals of validation, usability, and extendability, considerable care must be taken in the entire research enterprise.

> This includes not just the programming, but also the documentation and data analysis. (...) I have learned the hard way that haste does indeed make waste. Quick results are often unreliable. Good habits slow things down at first, but speed things up in the long run.

Debugging a program is always a difficult task and we can never be sure of producing completely error free code. The difficulty is worse in the context of agent based models, where the results of simulations can be unexpected and we cannot be sure whether they arise from features of the agents and their interaction, or from some hidden bug. It is therefore especially important to carefully check the code and to apply the simulation model to relatively well understood and predictable situations before exploring new territories.

As Axtell and Epstein (1994:31) point out:

> Such software problems are difficult to discover due precisely to the highly distributed nature of agent-based models. Indeed, the "robustness" of macro-structures to perturbations in individual agent performance - "graceful degradation" to borrow a phrase from neural networks - is often a property of agent based-models and exacerbates the problem of detecting "bugs."

Nevertheless, we can look for some "alarm" signals that can alert us to the presence of bugs. For example, if the simulation model shows greatly varying results when we change a single parameter *a priori* believed to be not critical, the first thought should be that the observed behaviour is due to a bug, not to some exotic emergent phenomenon. The problem becomes more subtle if the model is highly sensitive to initial conditions: unfortunately, this is frequently a highly desirable property especially if there are time series containing chaotic sequences. The best strategy to identify bugs is to examine carefully the internal data of the agents, to verify their exact behaviour. This is very easy with a high-level development library such as Swarm, where we can deploy probes pointing to any part of the code and examine values “on the fly”. The design of Swarm programs involves two different levels. There is the model level, where we can have nested models (or swarms), and the observer level which considers the model (or the nested models) as a unique object from which to extract the results using probes and send them on to various display tools and widgets.

4.3 Replicating simulated experiments

The use of standard agent based modelling tools and the introduction of general applications, which we can employ as “containers” based upon a specific shell, simplifies the task of replicating simulated experiments. There are several such tools available. The following are particularly suitable for building agent-based models, because they all implement object-oriented programming systems:

Lisp-Stat, http://stat.umn.edu/~luke/xls/xlsinfo/xlsinfo.html, is a tool providing a Lisp environment, statistical functions and easy to use graphics, such as histograms, scatterplots and spin-plots. In this object oriented environment agent models can be developed and the native capabilities of the language used to observe them; see, as an example, Gilbert (1999). Lisp-Stat is available for Macintosh, PC and Unix computers and works almost identically on each.

Swarm, http://www.santafe.edu/projects/swarm/, is a set of function libraries, based upon Objective C. The swarm, a combination of a collection of objects and a schedule of activity over those objects, is the basic building block of Swarm simulations. With Swarm, general purpose applications can be constructed, based on specific techniques, but easy to adapt to any specific problem. An example is the BP-CT neural network based application mentioned in Section 4.1.

MAML, Multi-Agent Modeling Language, at http://www.syslab.ceu.hu/maml. The goal of MAML is to provide the functionality of Swarm without the need to be familiar with Swarm's underlying low-level language, Objective-C. The current version of MAML defines a macro-language for Swarm. This introduces higher level concepts of modelling, and simplifies some constructs already included in Swarm by hiding the technical details needed to program them. However, the macro-language nature of MAML does not affect the user's access to the whole Swarm machinery. Its compiler (xmc) generates Swarm (v1.02 or v1.3) code that can be compiled again using the gcc compiler.

SDML, Strictly Declarative Modelling Language, http://www.cpm.mmu.ac.uk/sdml, has the following features: knowledge is represented in rulebases and databases; all knowledge is declarative; models can be constructed using many interacting agents; complex agents can be composed of simpler ones; object-oriented facilities, such as multiple inheritance, are provided; temporal facilities are provided, including different levels of time; rules can be fired using forward and backward chaining. SDML is implemented in Smalltalk.

LSD, Laboratory for Simulation Development, allows one to create, observe, edit and run micro-founded simulation models. It offers a set of tools that facilitate the most common operations for building and using simulation models, and particularly models with many nested entities as agent based ones usually are. Version 1.1 may be found at http://www.business.auc.dk/~mv/Lsd1.1/Intro.html .

SIM_AGENT, http://www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html, is an 'agent architecture' toolkit. It allows rapid prototyping and testing of all kinds of agents, from very simple reactive agents to those that can plan and deliberate. It is written in Pop-11 and has been used for experimenting with different kinds of agent architectures that meet a requirement for real-time response in complex and dynamic domains.

SimBioSys, is a general C++ class library for evolutionary simulation. Like Swarm, SimBioSys provides users with a general library aimed at building agent-based evolutionary models of socio-economic and biological processes. It is described by Leigh Tesfatsion, at http://www.econ.iastate.edu/tesfatsi/platroot.ps.

StarLogo, http://starlogo.www.media.mit.edu/people/starlogo/, is a programmable modelling environment for exploring the behaviour of decentralised systems (systems that are organised without an organiser, and co-ordinated without a co-ordinator). StarLogo has been used to model phenomena such as bird flocks, traffic jams, ant colonies and market economies. StarLogo was created for the Macintosh, but a Java version and a PC version are under development (the PC version actually exists, but it is an alpha version).

5 Conclusion

Computer simulation is set to become an important new method of building and evaluating theories in the social sciences. Like all new methodologies, it will take some time to refine the techniques and to codify them so that a minimum of trial and error is needed. At present, experimenting with computer simulations is still an art best learnt through practice and by closely observing those more experienced. In this paper, we have begun to set down some hard-won lessons from our own research, but there is much more to say, some of which can be gleaned from recent, methodologically inclined publications (e.g. Axelrod 1997b; Gilbert and Troitzsch 1999). Finally, good luck with your own creations!

6 References

Arthur, W. B. (1989) Competing technologies, increasing returns, and lock-in by historical events. *The Economic Journal*, 99:116-131.

Axelrod, R. (1997a), *The Complexity of Cooperation*, Princeton, Princeton University Press.

Axelrod, R. (1997b) Advancing the art of simulation in the social sciences. In R. Conte, R. Hegselmann and P. Terna (ed.) *Simulating social phenomena.* Berlin: Springer, pp. 21-40.

Axtell, R.L., Epstein, J.M. (1994), Agent Based Modeling: Understanding Our Creations, *The Bulletin of the Santa Fe Institute*, Winter 1994, pp.28-32.

Ballot, G. and Taymaz, E (1999) Technological change, learning and macro-economic co-ordination: an evolutionary model. *Journal of Artificial Societies and Social Simulation*, vol. 2, no. 2, <http://www.soc.surrey.ac.uk/JASSS/2/2/3.html>

Beltratti, A., Margarita S., Terna P. (1996) *Neural Networks for Economic and Financial Modelling.* London: ITCP.

Brately, P. Fox, B., and Schrage, L. E. (1987) *A guide to simulation.* 2nd edn. New York: Springer.

Chattoe, E. and N. Gilbert (1997) A simulation of adaptation mechanisms. In budgetary decision making' in R. Conte, R. Hegselmann and P. Terna (ed.) *Simulating social phenomena* Berlin: Springer, pp. 401-418.

Garson, G. D (1998) *Neural networks: an introductory guide for social scientists.* London: Sage.

Gilbert, N. (1995) Emergence in social simulation. In N. Gilbert and R. Conte (ed.) *Artificial Societies: the computer simulation of social life.* London: UCL Press, pp. 144-156.

Gilbert, N. (1999), Multi-level simulation in Lisp-Stat, *Journal of Artificial Societies and Social Simulation* vol. 2, no. 1, <http://www.soc.surrey.ac.uk/JASSS/2/1/3.html>.

Gilbert, N and Troitzsch, K.G. (1999) *Simulation for the Social Scientist*. Milton Keynes: Open University Press.

Hegselmann, R. (1996) Understanding social dynamics: the cellular automata approach. In K. G. Troitzsch, U. Mueller, G. N. Gilbert and J. E. Doran (ed.) *Social Science Microsimulation* Berlin: Springer, pp. 282-306.

Hegselmann, R. (1998) Modelling social dynamics by cellular automata. In W.B.G. Liebrand, A. Novak and R. Hegselmann (eds.) Computer modeling of social processes. London: Sage.

Holland, J. H and Miller, J.H (1991) Artificial adaptive agents in economic theory. *American Economic Review* 81(2), pp. 365-370

Hutchins, E. and B. Hazlehurst (1995) How to invent a lexicon: the development of shared symbols in interaction. In G. N. Gilbert and R. Conte (ed.) *Artificial Societies*. London: UCL Press.

Kleijnen, J.P.C. (1998) Validation of Simulation, With or Without Real Data, *Department of Information Systems and Auditing (BIKE)/Center for Economic Research (CentER) Working Paper*, home page <http://cwis.kub.nl/~few5/center/staff/kleijnen/cv2.htm>.

Klüver, J. (1998) Modelling science as an adaptive and self-organising social system: concepts, theories and modelling tools. In P. Ahrweiler and N. Gilbert (eds.) *Computer simulations in science and technology studies*. Berlin: Springer.

Moss, S., Gaylard, H., Wallis, S. and Edmonds, B. (1988), SMDL: a multi-agent language for organizational modelling. *Computational and mathematical organization theory*, 4(1), pp. 43-70.

Nowak, A. and B. Latané (1994) Simulating the emergence of social order from individual behaviour. In N. Gilbert and J. Doran (ed.) *Simulating Societies: the computer simulation of social phenomena* London: UCL Press, pp. 63-84.

Ostrom, Thomas (1988) Computer simulation: the third symbol system. *Journal of Experimental Social Psychology*, 24:381-392.

Swarm, http://www.santafe.edu/projects/swarm/

Terna, P (1997) A laboratory for agent based computational economics. In Conte, R., R. Hegselmann and P. Terna, Ed. (1997) *Simulating social phenomena.* Berlin: Springer.

Terna, P. (1998a), Simulation Tools for Social Scientists: Building Agent Based Models with SWARM. *Journal of Artificial Societies and Social Simulation* vol. 1, no. 2, <http://www.soc.surrey.ac.uk/JASSS/1/2/4.html>.

Terna, P. (1998b), ABCDE: Agent Based Chaotic Dynamic Emergence. In J.Sichman, R.Conte and N.Gilbert (eds), *Multi-Agent Systems and Agent-Based Simulation*, LNAI series, vol. 1534, Berlin, Springer-Verlag.

Verhagen, H. (1999) ACTS in action: Sim-ACTS - a simulation model based on ACTS theory. In J. Sichman, R. Conte and N. Gilbert (Eds), *Multi-agent systems and agent-based simulation*. Berlin: Springer, pp. 1999-209.

Wooldridge, M. and N. R. Jennings (1995) Intelligent agents: theory and practice. *Knowledge Engineering Review* vol. 10(2): 115-152.

Woolgar, S. (ed.) (1988) *Knowledge and reflexivity: new frontiers in the sociology of knowledge*. London: Sage.

Zeigler, B.P. (1985) *Theory of modelling and simulation*. New York: Wiley.