A Lifecycle for Models of Large Multi-agent Systems*

Wamberto Vasconcelos[1,a], David Robertson[1,b], Jaume Agustí[2,c], Carles Sierra[2,d], Michael Wooldridge[3,e], Simon Parsons[3,f], Christopher Walton[1,g], and Jordi Sabater[2,h]

[1] Division of Informatics, University of Edinburgh, Edinburgh EH1 1HN, UK
{[a]wamb, [b]dr, [g]cdw}@dai.ed.ac.uk
[2] Institut d'Investigació en Intel·ligència Artificial (IIIA)
Campus UAB, 08193 Bellaterra, Catalonia, Spain
{[c]agusti, [d]sierra, [h]jsabater}@iiia.csic.es
[3] Department of Computer Science, University of Liverpool, Liverpool L69 7ZF, UK
{[e]M.J.Wooldridge, [f]S.D.Parsons}@csc.liv.ac.uk

Abstract. Two key issues in building multi-agent systems concern their *scalability* and engineering *open systems*. We offer solutions to these potential problems by introducing a lifecycle for models of large multi-agent systems. Our proposal connects a model for the collective analysis of agent systems with an individual-based model. This approach leads on to a virtuous cycle in which individual behaviours can be mapped on to global models and vice-versa. We illustrate our approach with a formal example but relatively easy for engineers to follow and adapt.

1 Introduction

A key issue in building multi-agent systems (MASs) is that of scalability. Large MASs are globally distributed intercommunicating collections of human, software and hardware systems each with hundreds or thousands of components. Such systems are becoming increasingly common and complex. Ensuring that these systems do not fall into unstable or chaotic behaviour is a main challenge designers face.

The wealth of knowledge about conventional information processing does not seem to scale up nor adapt easily to MASs [8,9]. It is important that we study and understand the dynamics of MASs, the forces that define these dynamics, and how these can benefit or impair a MAS. Hopefully this knowledge should cast light on issues of design, performance and reliability of components of MASs.

Another key issue concerns engineering open systems. A canonical example is the Internet: with the predicted increase in the number of personal agents (information filtering and shopping agents being two widely studied examples) and commercial agents (shopbots and pricebots, for example) the Internet seems likely to become an environment teeming with agents encountering situations for which they were not explicitly designed.

* Work sponsored by the European Union, contract IST-1999-10208, research grant **Sustainable Lifecycles in Information Ecosystems** (SLIE).

M.J. Wooldridge, G. Weiß, and P. Ciancarini (Eds.): AOSE 2001, LNCS 2222, pp. 297–317, 2002.

At present we lack a theoretical understanding of how to design systems which can be guaranteed to exhibit good performance in such a free-for-all, and those empirical studies that have been carried out suggest that system behaviour is likely to be extremely complex (*e.g.* [10,11]). In contrast, as can be seen in other papers in this volume, we already have a range of tools and techniques for engineering closed agent systems.

We can model very large MASs at different levels of abstraction. At a very high level of abstraction we want to study the aggregate behaviour of agent populations. For this we suggest the use of equations of continuous change, abstracting away from the details of individual interaction. Such collective models can then be used to drive more detailed modelling of individual agent interactions: the global behaviour from our continuous models helps to map the behaviours we examine in more detail when we devise an agent-based modelling. The issue is then to build models of interaction at the level of individual agents which are as small and simple as possible while allowing us to prove or disprove hypotheses about the relation between agent structure (at the individual agent level) and MAS behaviour (at the population level). We demonstrate by example how this can be done using a method which is formal but is relatively easy for engineers to apply.

In the next section we describe the example problem we shall use to describe our approach. In Sections 3 and 4 we give more details of our proposed method for the study of MASs. In our approach, we employ established analysis techniques from dynamic systems, but we also use agent-based simulation. The two apparently conflicting views are, in fact, means to exploit different and complementing aspects of MASs. Section 5 presents a lifecycle embodying the two steps of our previous sections. In Section 6 we draw conclusions and give directions for future work.

Our interest in this topic stems from our involvement in the research grant **Sustainable Lifecycles in Information Ecosystems** (SLIE) [17]. The main goal of this project is to develop robust methodologies for the design, analysis, deployment and maintenance of successful components of MASs.

2 Example Problem

Our example concerns the distribution of resources in those supply systems for which it is important to ensure some uniform distribution of resources amongst a number of agents. If such a system has an uneven distribution of resources, we would like to know what sorts of behaviours this might engender and whether these behaviours are likely to differ as the size of our system increases.

To perform coherent sets of experiments we must limit the variability of the systems under study so we make the following idealisation assumptions in our modelling:

1. Resources are neither created nor destroyed by any agent so the only issue is transfer of resources from one agent to another. This rules out influences of internal resource generation or import of resources.
2. Transfer of resource is "frictionless". There is no loss of resource in transit; impediment to the amount of resource transferred; or disparity between the amount of resource one expects to have and the amount held.

3. All agents have the same decision procedure for choosing how much resource to offer or request when trading.
4. Every transfer of resource involves a balance between the needs of the donor and recipient so, for example, it is not possible for any agent to receive resource beyond what it calculates it needs.

These are strong assumptions which allow us to build models of an ideal case. This ideal case is valuable because it provides a baseline – a system which is unstable under the ideal assumptions is likely to be even more unstable if the assumptions are weakened. We do not expect the ideal case itself to occur in real systems but we expect real systems to have at least as many problems as we find in the ideal case.

3 Equation-Based Modelling (EBM)

Standard techniques and tools for dynamic systems analysis, modelling and simulation [4,6] are used in this first step. Dynamic systems have been successfully modelled and simulated by means of equations [6]. Standard techniques coupled with practical tools make dynamic systems modelling and simulation a straightforward engineering effort. We intend to build on this state-of-the-art, our initial model being equation-based.

We wish to relate behaviours hypothesised at the population level to features at the individual level. The first modelling step is therefore to develop our hypotheses about population behaviours. We do this by building a model which describes the essence of the dynamics, at a population level, of our idealised system. We hope that this model (although itself containing no agents) will generate interesting behaviours which we can then seek to understand in terms of agent structure using more detailed modelling. To be controllable and easily analysed, we require this early model to be as simple as possible while still generating insights into potential behaviours.

The model we use at this stage is expressed in a system dynamics modelling style [4,6]. We imagine the agent system to be composed of some number of sub-populations (the number of agents in each of these being immaterial) with the quantity of resource held by each sub-population being described by a single, integer state variable. Resource flow between each sub-population is controlled by a rate variable associated with that flow. Formally, this can be described by the following equations:

$$\begin{aligned} state_v(X, initial\_time) &= initial\_value(X) \\ state_v(X, T) &= state_v(X, previous(T)) + \\ &\quad \sum\{rate_v(F_i, previous(T)) | \exists X_i . flow(F_i, X_i, X)\} - \\ &\quad \sum\{rate_v(F_o, previous(T)) | \exists X_o . flow(F_o, X, X_o)\} \end{aligned}$$

where $state_v(X, T)$ is the value for the state variable X at time point T; $rate_v(F, T)$ is the value for the rate variable of flow F at time point T; $initial\_time$ is the initial time point in the simulation; $initial\_value(X)$ is the initial value for state variable X; $previous(T)$ is the time point previous to T; and $flow(F, X_i, X_o)$ is true if there is a flow named F from state variable X_i to state variable X_o.

The definition above is generic for this sort of continuous flow model. We now decide on the specific model topology needed for our example. Many topologies are possible. For instance, we could have a fully connected distribution network in which all sub-populations are connected by flows to each other or we could select randomly which sub-populations are connected. One of the simplest topologies for our purposes is a distribution ring, in which we have a chain of sub-populations with the last link of the chain connecting back to the first. This topology allows a simple measure of size of system: the number of sub-populations in the ring, which we shall call the ring size and shall write as $\mathcal{R}$. It also allows a simple way of allocating an uneven distribution of resources to sub-populations at the start of the simulation: the total amount of resource for the population is distributed in proportion to its position in the ring. Some examples of distribution ring topologies generated in this way are shown in Fig. 1.

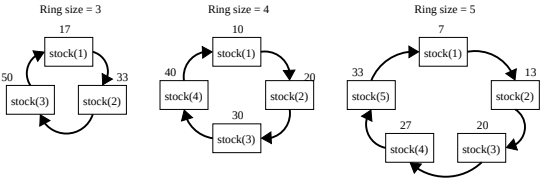


Fig. 1. Possible Distribution Ring Topologies

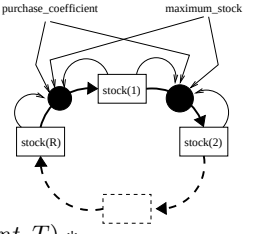
Our entire model is now a function of the ring size ($\mathcal{R}$). By setting $\mathcal{R}$ to different values we generate models of different sizes and resource distributions but with a uniform interaction between adjoining sub-populations in the ring. The formal definitions needed to achieve this sort of parameterisable model are shown in Fig. 2.

We are now in a position to run the model. We run a simulation for 1000 time steps for each value for $\mathcal{R}$ from 3 to 10. This simulation can be run with any system dynamics modelling package or it can (for uniformity with our other specifications) be run using a Prolog meta-interpreter. The meta-interpreter we used for this example can be obtained on-line in [17].

Fig. 3 shows a plot of the resource held by the first sub-population of the ring[1] for each value of $\mathcal{R}$. The lines for lowest ring sizes are those which start at the highest resource level. We can see from Fig. 3 that the time taken to obtain a stable distribution of resources increases, probably exponentially, with ring size (for $\mathcal{R} = 3$ stability is around time 180; for $\mathcal{R} = 4$ stability is around time 450; for $\mathcal{R} = 5$ stability is just beyond time 1000). Furthermore, at larger ring sizes

[1] Notice that we must focus on one of the sub-populations rather than taking a mean for the whole ring because the total resource for the ring remains constant throughout.

$flow(transfer(N1, N2), stock(N1), stock(N2)) \leftarrow ring(\mathcal{R}, N1, N2)$
where $ring(\mathcal{R}, N1, N2)$ is true when $N1$ and $N2$ are adjoining ring elements.



$$initial\_time = 1$$
$$parameter(purchase\_coefficient) = 0.2$$
$$parameter(maximum\_stock) = 100$$
$$previous(T) = T - 1$$
$$initial\_value(stock(N)) = \mathcal{R} * \frac{parameter(maximum\_stock)}{\mathcal{R}!}$$
$$rate_v(transfer(N1, N2), T) = parameter(purchase\_coefficient, T) * state_v(stock(N2), T) * 1 - \frac{(parameter(maximum\_stock) - state_v(stock(N1), T)}{parameter(maximum\_stock)} * 1 - \frac{state_v(stock(N2), T)}{parameter(maximum\_stock)}$$

Fig. 2. Definitions of Continuous Change for Resource Ring of Size $\mathcal{R}$

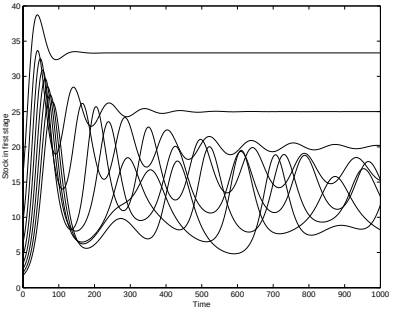


Fig. 3. Behaviours of Continuous Change Model

we find sustained and pronounced oscillations which become more complex as we increase the ring size.

These results show that we can quite rapidly obtain oscillatory behaviour in resource distribution, even when we have a highly uniform distribution structure. This is the case for systems which are sufficiently large so that the transactions between agents can be viewed as continuous flows of resource between sub-populations. Since the time taken for these oscillations to dampen to a steady state grows very quickly with increased system size, it would be possible for the oscillations in a large system to persist beyond its life span. The form of these oscillations may not be simple and easily predictable for large systems so it would be possible to have this sort of behaviour within a system without being aware, when looking at a sample of the time series, that a real oscillation was at work.

4 Agent-Based Modelling (ABM)

What features of individual agent structure could account for the increased delay in stabilisation with larger population sizes which we modelled in the previous section? As before, we want a parsimonious agent model (the simplest which will exhibit this sort of behaviour). However, parsimony must now be obtained with a different style of modelling in which the elements of the model are individual agents and communication between individuals is through (possibly concurrent) message passing.

When specifying our agent model we are not as constrained in our style of specification as we might be for specifications of agents which are to be deployed in real systems. In particular, we are free to specify in a straightforward way information about the global social structure of agents systems. For this we use a simplified form of the concept of *electronic institutions* [12,14,18], explained in Section 4.1 and put to use in Section 4.2. This gives us a way of expressing the agent system at the level of the potential interactions between agents, without specifying exactly how these interactions occur. The specification of potential interactions then gives us starting points for defining the decision procedures of types of individual agents, as we show in Section 4.3.

4.1 Electronic Institutions

The scenario in which the agents will perform should be formally described. For this we use *electronic institutions* [12,14,18]. These are formal structures based on process algebras that enable the specification of conventions and norms of individual and collective behaviour in a community of interacting agents. Some notions of an electronic institution are:

- *Agents and Roles* – agents are the players in an electronic institution, interacting by the exchange of illocutions, whereas roles are defined as standardised patterns of behaviour. The identification and regulation of roles is considered as part of the formalisation process of any organisation. Any agent within an electronic institution is required to adopt some role(s). As actions are associated to roles, an agent adopting a given role is allowed to perform the actions associated to that role. A major advantage of using roles is that they can be updated without having to update the actions for every agent on an individual basis.
- *Scene* – interactions between agents are articulated through agent group meetings, which we call scenes, with a communication protocol. We consider the protocol of a scene the possible dialogues agents may have.
- *Performative Structures* – scenes can be connected, composing a network of scenes (the so-called performative structure) which captures the existing relationships among scenes. The specification of a performative structure contains a description of how the different roles can legally move from scene to scene. A performative structure is to contain the multiple, simultaneous ongoing activities, represented by scenes. Agents in a performative structure may take part in different scenes at the same time with different roles.

– *Normative Rules* – agent actions in the context of an institution may have consequences that either limit or enlarge its subsequent acting possibilities. Such consequences will impose obligations to the agents and affect their possible paths within the performative structure.

Sophisticated interactions among agents can be elegantly expressed via institutions. These can be seen as finite-state machines describing all the possible message exchanges that may take place in a certain situation. We have employed an implementation of electronic institutions in our specification and simulation of MAS described below.

4.2 Specification at Agent Interaction Level

To specify potential agent interactions we use a notation similar to a simplified form of CCS (Calculus of Communicating Systems, [13]) but with a syntax adapted to agent modelling. In this section we describe the language and show how it is applied to our running example. An interaction specification is a set of clauses, each of the form A `::=` D, where:

– A is an agent identifier of the form `agent(`S, R, A`)`, where S is the scene to which the agent belongs; R is the agent's role in that scene; and A is the unique name of that agent.
– D can be of the form:
  • D_1 `par` D_2 denoting that D_1 and D_2 occur in parallel.
  • D_1 `then` D_2 denoting that D_1 must occur before D_2 (*i.e.* sequence).
  • D_1 `or` D_2 denoting that D_1 or D_2 can occur but not both (*i.e.* choice).
  • M `=>` A_e where M is a message sent to an external agent A_e.
  • M `<=` A_e where M is a message received from an external agent A_e.
  • `agent(`S', R', A`)` where S' is a scene (possibly different from S) and R' is the agent's role in the scene (possibly different from R).

With this language we can describe a system of scenes for interaction between agents, stipulate the roles played by agents in these scenes, and define the ordering of interactions (through message passing) for different types of agent depending on which role and scene they inhabit. Our running example concerns resource distribution and it is common for this type of agent system to involve three sorts of scene: an agora in which agents with resource to buy or sell are put in touch with each other by a broker; a negotiation scene in which agents which were paired together in the agora interact to agree on a contract to transfer resource; and a delivery scene in which contracts to supply resource are either honoured or defaulted upon. Below we specify these in detail.

A broker agent in the agora can receive an offer of stock and request for stock (in parallel) from two traders in the agora; then may inform the trader offering stock of the name of a potential buyer; then may inform the trader requesting stock of the name of a potential supplier. Then it continues as a broker in the agora. Formally:

```
agent(agora,broker,B) ::=
    ( offer(stock,_) <= agent(agora,trader,A1) par
```

```
      request(stock,_) <= agent(agora,trader,A2) par
      (offer(buyer(A2),_) => agent(agora,trader,A1) then
       offer(seller(A1),_) => agent(agora,trader,A2) ) ) then
    agent(agora,broker,B).
```

A trader in the agora can offer stock to a broker and receive notice of a buyer from the same broker, then it becomes a supplier in the negotiation scene, or it can send a broker a request for stock and receive notice of a supplier, then it becomes a customer in the negotiation scene:

```
agent(agora,trader,X) ::=
    ( offer(stock,_) => agent(agora,broker,B1) then
      offer(buyer(_),_) <= agent(agora,broker,B1) then
      agent(negotiation,supplier,X) ) or
    ( request(stock,_) => agent(agora,broker,B2) then
      offer(seller(_),_) <= agent(agora,broker,B2) then
      agent(negotiation,customer,X) ).
```

The remaining cases are depicted in Appendix A. We now use this as a framework for introducing the definitions of individual agents.

4.3 Specification at Agent Decision Level

The decision procedures for agents are expressed using two types of clause. The first type defines the conditions under which a message can be sent and the second stipulates the reaction when a message is received. The syntax of these is, respectively, as follows:

$$A \texttt{ ::= } M_r \texttt{ <-- } C_1 \texttt{ and } C_2 \texttt{ and } \ldots \texttt{ and } C_n$$
$$A \texttt{ ::= } M_s \texttt{ --> } R_1 \texttt{ and } R_2 \texttt{ and } \ldots \texttt{ and } R_n$$

where A is the identifier for an agent; M_s is a message sent to that agent; M_r is a message received by an agent; C_i are conditions which can be proved true in the agent and R_i are conditions which can be made true in the agent. In the remainder of this section we show how clauses of this form can be used to describe detailed decision procedures for the types of agent in our running example.

A trader in the agora can send an offer of stock to a broker if it knows the name of the broker and it has a surplus of stock above 50 units. A trader in the agora can send a request for stock to a broker if it knows the name of the broker and it has a deficit of stock below 50 units. More formally:

```
agent(agora,trader,_) ::=
  offer(stock,N) => agent(agora,broker,B) <--
    broker(B) and resource(stock,NS) and N is NS - 50 and N > 0.
agent(agora,trader,_) ::=
  request(stock,N) => agent(agora,broker,B) <--
    broker(B) and resource(stock,NS) and N is 50 - NS and N > 0.
```

The remaining cases are depicted in Appendix B.

4.4 Simulation Directly from Specification

We now have enough detail in the specification, both in possible interactions and in individual decision procedures, to run a simulation of the system. This could be done by translation to an execution language of choice or also by executing our specification directly, via a meta-interpreter.

Our specifications can be executed either sequentially or concurrently. In the former case, the meta-interpreter takes turns executing portions of each agent until it cannot progress due to an interaction (via message-passing) with another agent; the meta-interpreter then chooses another agent to execute portions of and so. In the concurrent execution, copies of the meta-interpreter are started up simultaneously each with the specification of the agent it should execute – the agents are independently executed by their meta-interpreter, provided adequate message-passing services are offered.

Ours is a sequential execution/simulation of our specification that can replicate runs – concurrent executions may be very difficult to re-enact when trying to debug or understand phenomena that took place. The sequential meta-interpreter, written in Prolog, we developed and employed can be obtained in [17]. It works as follows:

1. We supply the interpreter with a list of agents in our population and conditions for terminating the simulation.
2. Each agent in the list is replaced with a matching agent definition from the interaction clauses of Section 4.2.
3. The interpreter then attempts to expand appropriate subterms in the agent definitions according to the position of each subterm. For example if the definition is of the form A `then` B, the subterm A will be expanded before the subterm B.
4. Subterms corresponding to sending messages are expanded only when one of the decision procedure clauses of Section 4.3 permits it. The resulting message appears in the message queue.
5. Subterms corresponding to receiving messages are expanded only if a matching message is in the message queue. An appropriate reaction clause from Section 4.3 is matched and its consequences are made true in the agent.
6. This continues until no more rewrites of the current term are possible (in which case the interpreter may backtrack to search for alternatives) or the conditions for termination are satisfied.

The term constructed by this rewriting system represents the history of interaction in the simulation. An example of this sort of term is shown in Appendix C. This was produced for the simplest possible trading system, consisting of two traders ($a1$ and $a2$) and a broker. The appendix shows the formal structure of the constructed term and, in Fig. 4, we present the interactions which took place between agents. We can also analyse the interaction level of our specification for desirable/unwanted properties using standard model-checking techniques [1].

We now are in a position to return to the question with which we began Section 4: what features of individual agent structure could account for the increased delay in stabilisation with larger population sizes which we modelled in the previous section? Specifically, does the agent model we have just defined have

Fig. 4. Graphic Representation of Interactions in the 2-Agent Model

this sort of behaviour? To test this we run the model with increasing numbers of agents and measure the CPU time taken to obtain a uniform resource distribution (at a level of 50 units) and the number of rewrite steps contained in the final interaction term. The former is a measure of the amount of search needed to obtain a uniform distribution; the latter is a measure of the complexity of the distribution task. The table below shows results for models with 2, 4 and 6 trading agents:

Number of Traders	CPU Time	Rewrite Steps
2	70	19
4	220	40
6	6570	61

This shows that the number of rewrite steps needed to reach a stable resource distribution does not rise sharply with increasing numbers of trading agents but the CPU time taken to find the appropriate combination of rewrites rises very sharply. Most of the cost in CPU time for larger systems is in backtracking: when the simulator has to unravel a sequence of message passing which did not lead to a stable resource distribution. In other words, quite simple solutions to resource distribution problems may exist in this sort of agent model but simple negotiation strategies can take a very long time to discover them and are very likely to make local commitments which are sub-optimal for the global system.

5 A Lifecycle for Models of Large MASs

The basic idea grounding our methodology is that the two modelling views introduced in the previous sections, *i.e.* EBM and ABM, correspond to subsequent specification steps. We take the stance that in order to build a model for a society containing thousands or millions of agents, the general view provided by an EBM provides succint descriptions of population-level behaviours which we then attempt to replicate using models consisting of individual interacting agents, that is, the ABM. When dealing with a small number of interacting agents a different approach could be taken (*e.g.* [22]). Our proposed lifecycle is graphically depicted in Fig. 5. We note here that the focus of our attention is

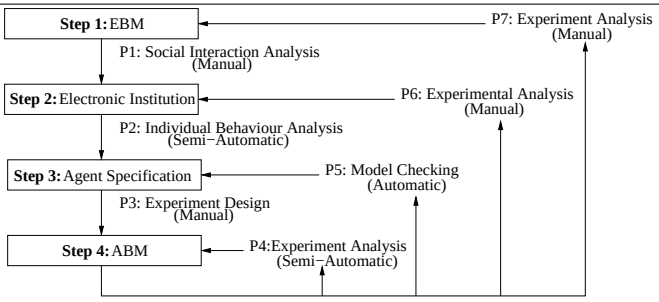


Fig. 5. A Lifecycle for Models of Large MASs

the *experimentation* of the interaction of groups of agents in a controlled environment with the general objective of giving insights for the design of MASs and not the *deployment* of stable agents.

An important characteristic of MASs from a software engineering perspective is the decoupling of two aspects: the interaction process between agents from the deliberative/reactive activity within each agent. This decoupling helps in simplifying the development of complex software systems and has guided methodological approaches [3,23]. We understand interaction as constrained by sets of norms that are enforced by an intermediation infrastructure [14,19,15,18]. The notion of *electronic institution*, as described in Section 4.1, plays this role in our methodology by establishing a framework that constraints and enforces the acceptable behaviour of agents. The different phases are:

Step 1 **EBM – Equation-Based Model**. In this first step, a set of state variables and equations relating them must be identified. These equations have to model the desired global behaviour of the agent society and will not contain references to individuals of that society. Typically these variables will refer to values in the environment and to averages of predictions for observable variables of the agents. Methodologically speaking, our approach has an essential difference with respect to the natural and social sciences that model their systems using EBMs. All other fields model *existing* systems: ecosystems, economies, physical systems, and so on. We, instead, are modelling yet-to-exist artificial systems. This distinction is crucial as the EBM is the starting point of the construction of a system that later on – once completely constructed – will be observed. Thus, a comparison between the EBM predicted behaviour and the actual ABM behaviour will be obtained.

Step 2 **EIM – Electronic Institution Model**. In this step the possible interactions among agents are the focus. It is a first "zoom in" of the methodology from the global view towards the individual models. The EBM obtained in the previous step determines different relations between environment variables and the agent society observables. These relations will now guide the building of an electronic institution permit-

ting interactions between agents supported by the EBM. This step is not a refinement of the EBM but rather the design of a set of social interaction norms that are consistent with the relations established at Step 1. For instance, if in the EBM we model a global transfer of resources from a population of agents to another population of agents we will need to establish the means in the institution for the individual agents of these populations to interact for the individual transfers (aggregated in a variable in the EBM) to happen. Nonetheless, this is not enough since the designer will have to make new decisions at this stage, such as what type of interaction, what norms will determine each interaction and whether there are going to be contracts between agents, under which circumstances they will interact and so on. Additionally, constraints on the individual behaviours have to be established.

Step 3 **ABM – Agent-Based Model**. Here, we focus in the individual. We have to decide what decision models to use. This is the second "zoom in" of the methodology. Once the interaction conditions have been established, it is time in the methodology to design different agent models that will show an external behaviour that respects the interaction protocols and norms established in the EIM. For instance, when designing a "producer", the concrete strategy to interact – negotiate – with "consumer" agents, has to be decided. All admissible strategies could be explored at this stage – in principle, all strategies compatible with the EIM and EBM specified before. New elements of the requirement analysis (new variables) will be taken into account here. For instance, some rationality principles associated to agents (*e.g.* producers do not sell below production costs), or negotiation models to be used (*e.g.* as those proposed in [20]) have to be selected.

Step 4 **Multi-Agent System**. Finally, the last step of our methodology consists on the design of experiments for the interaction of very large numbers of agents designed in the previous step. For each type of agent the number of individuals and the concrete setting for the parameters will be the matter of decision here. The results of these experiments will determine whether the requirements of the artificial society so constructed have been consistently interpreted throughout the methodology and thus whether the expected results according to the EBM are confirmed or not. Moreover, our methodology permits the developer to establish at this step sets of formal and verifiable claims about the expected results and properties of the agents interaction in that particular experiment.

The different stages of the methodology are to be traversed from Step 1 to Step 4, that is, from the desired global behaviour down to the experiments. However, once the experiments designed at Step 4 are run and analysed, several redesigns are possible. Next, we enumerate the different feedback processes of the methodology:

P1 **Social Interaction Analysis**. Once the EBM has been constructed, the relations between the global variables and the analysis of the requirements of the society to model will determine what sort of agents exist (*i.e.* the roles), what sort of interactions the agents must have (*i.e.* the scenes), and

what sort of transactions or dialogues they will have (*i.e.* ontology). This is an inherently manual process: there are many decisions to be made at this stage that have not been specified in the EBM. Experience in designing electronic institutions will help in deciding what scenes are most appropriate.

P2 **Individual Behaviour Analysis**. Once a complete picture of the institution is ready, the final aspect to consider is the modelling of the behaviour of the agents. Many aspects of this behaviour are already determined by the institution. The performative structure, the protocol of the scenes and the ontology definition enormously limit the repertoire of the decisions to be made by the agents [12,14,18], hence in many cases the behaviour is almost completely determined. For those aspects that are not completely determined the methodology strongly encourages the design of parametric decision models to fill in the gaps. These parameters will be used to set different experiments and will be the target of the agent design rules.

P3 **Experiment Design**. By choosing agents to participate with (possibly) different decision mechanisms, and by giving concrete values to the parameters of those decision mechanisms, different experiments can be constructed. The experiments should be set so as to explore all the possibilities and to see whether the EBM is making the right prognosis.

P4 **Experiment Analysis (ABM redesign)**. The analysis of the experiments will be done by comparing the predicted values of the global variables by the EBM and the actual values of agent variables and their averages. Deviations on the values will be corrected at this stage by changes in the experiment design. That is, agent-based model parameter values will be changed. This phase will be semi-automatic, and we shall provide certain design rules that will determine what changes are to be made depending on the deviation detected, the current parameter value and the environment settings.

P5 **Model Checking**. The claims about the behaviour of a group of agents that the developer establishes when specifying an experiment will be model-checked [1] at this stage. The outcome of the model checking will help to change the agent-based models, *i.e.* change the decision-making models.

P6 **Experiment Analysis (EIM redesign)**. Additionally, when the model checking determines that certain properties can never be guaranteed or that after several trials it is impossible to find parameter values that lead to the expected correct behaviour, different constraints over the agents interactions could be explored. This means that a redesign of the EIM may be in place. This is an intrinsically manual task.

P7 **Experiment Analysis (EBM redesign)**. Finally, and if everything fails, it may happen that the part of the requirements that led to the initial EBM was misunderstood and that a variation in the initial EBM is necessary to explain why the experiments are showing a unexpected behaviours.

6 Conclusions and Directions of Research

Our proposal employs seemingly disparate approaches, equation-based (EBM) and agent-based modelling (ABM), to the task of modelling and analysing large MASs. System dynamics and EBM practitioners have a long-standing tradition

with a successful record in modelling complex systems and analysing their results. By using EBMs as our initial step, we build on this success. ABM, on the other hand, has not been established as a rigorous practice and success may not be directly transferred from one experience to another because of the lack of standards and methodology.

The issue of scalability has been explicitly addressed by providing a means of modelling system behaviour "in the large", that is at a high level of abstraction at which instabilities in behaviour can be detected before the detailed design of the agents is carried out. This is in agreement with conventional wisdom in software engineering which advocates a top-down approach, on the grounds that it is easier/cheaper to identify/fix such problems before the detailed design of individual components has been carried out. The issue concerning the engineering of open systems has also been dealt with: we have made use of techniques for structuring the space of possible interactions within a MAS, creating areas in which interactions can be controlled, while leaving agents free to choose which areas they want to interact in. Since we can predict the interactions within the controlled areas, this allows us to ensure suitable system behaviour, while giving agents the freedom to choose where to interact ensures that the system is not closed.

EBM provides us with a technique for describing aspects of a MAS at a very abstract level, and then exploring its behaviour. This gives the designer of such a system a tool for ensuring that the overall shape of the system is within its design parameters. It provides a means of checking for stability, identifying if chaotic behaviour will occur, and ensuring that certain classes of agents can find a suitable niche to survive (or indeed will not find a niche if that is a desirable outcome). Thus equational modelling is a means of handling the scalability issue, by allowing some experimental guarantees that the MAS will behave as expected well in advance of its actual construction. When an EBM is built and tested, it provides a first, high level, specification of the system. However, EBMs concentrate on collective behaviour and interesting phenomena may only take place on an individual level. Negotiation [16] and argumentation [21], for instance, are two examples of interactions that are essentially and intrinsically performed among individuals. Agent-based modelling is called in as a means to address phenomena arising from the behaviour of individuals.

After the equational model has been developed, the next stage is to design the relevant electronic institutions. Once the institutions have been identified, the sets of actions they permit and the roles of the agents within them have been specified, they then place conditions on the agents which can use them. Thus, from the specification of an institution we can derive a partially-instantiated specification of an agent which can use that institution. This can then be fleshed out with what we might consider the "personality" of the agent, that is, the way that it operates within the rules of the institution. This refinement of the design of individual agents will be carried out in exactly the same way as for existing agent-oriented software engineering techniques.

We have devised a preliminary formalism and the underlying machinery for the specification of ABMs and their simulation. An appropriate interface for users to interact with experimental scenarios, that is, customising and changing parameters of agents, is under way. The specification of electronic institu-

tions which will help define our experiments is a difficult and error-prone task. ISLANDER, a language to define institutions (and with which one can prove properties about it) has been defined with an associated visual tool to help the design (and test) of institutions [2,7].

Our experiments demonstrate fundamental limitations of size in multi-agent resource distribution systems. The basic structural problem is that negotiation strategies operate locally between agents, which seek individually to obtain resource stability. At a local level these choices may be rational. However, as we increase the size of the system the chances of a local decision being optimal with respect to global resource stability diminish. The time taken to reach stability tends to increase exponentially, as we saw in the agent model of Section 4. Furthermore, it may not be possible to know at any given point in the evolution of such a system whether or not we are converging on a stable distribution, since our experiments with a continuous flow model in Section 3 demonstrate that systems of very large size may produce complex patterns of oscillation. These oscillations, although repeating on a long time scale, may appear chaotic if observed only during a short segment of time.

There are two things to note here. The first is that these simulations are just another means of checking the design, one that operates at a lower level of abstraction than the equational model. We are not aiming to be able to directly execute the agents in a MAS. The entities in a simulation will be abstractions of the agents in the MAS itself. The second thing to note is that the simulations have an important complementary role to play alongside model checking. While model checking is to some extent a stronger way of verifying the behaviour of the system, in the sense that it is possible to prove that it behaves in a certain way, model checking itself is sufficiently computationally expensive that it is unlikely to be possible to model check an entire MAS at anything other than quite a high level of abstraction. But settling for the weaker guarantees offered by simulation, we can get results for a description of the MAS at a much lower level of abstraction.

Our proposed lifecycle for models of large MASs consists of constructive steps with connecting procedures/processes. Starting from an EBM, and following a sequence of well-defined processes, distinct stages/steps are reached with intermediate results that lead on to the final ABM. Some of the processes contemplate the analysis of such intermediate results with a view to correct or improve them (feedback). Ideally the activities of a lifecycle should be automatic, however in our proposal we have come across tasks that seem to be intrinsically manual, such as analysing the behaviours of an experimental electronic institution or the behaviours of an EBM.

The agents in our experimental framework have a clear distinction between their behaviour (as specified by the institutions they follow) and their decision procedures, that is, how agents decide on possible behaviours prescribed within an institution. This division is a profitable one: the behaviours of an institution can be proved/checked for desirable properties, and later run with different agents. The theoretical underpinning of institutions are finite state machines which enjoy desirable model-checking properties. Different languages can be used to define the decision procedures of agents. We have been experimenting with MABLE [24] a procedural language enriched with constructs from the

agent-oriented programming paradigm. A MABLE system contains a number of *agents*, each of which has a *mental state* consisting of beliefs, desires and intentions; mental states may be nested, so that (for example), one agent is able have beliefs about another agent's intentions. MABLE agents are able to communicate with one-another using performatives in the style of the FIPA agent communication language [5].

Having developed a way of formally describing institutions in the language ISLANDER, we can do more than just identify the conditions that participation in an institution places upon agents. We can actually generate a partial specification of an agent automatically from the institution specification and the role (or roles) that the agent will play in the institution. Currently we translate the ISLANDER specifications into MABLE. The reason for doing this is that MABLE programs can be model checked. Thus we can take the specification of an institution and the conditions it imposes on agents that will use it and compile this into a partial MABLE program. It will be partial because the "personality" of the individual agents will be missing. Once these are added, it is possible to model check the resulting program, formally verifying the behaviour of the institution and the agents which will make use of it, allowing the system designer to check that undesirable conditions like deadlocks do not occur. If they do, then either the institution or the agent "personalities" need to be altered.

References

1. M. Benerecetti, F. Giunchiglia, and L. Serafini. Model Checking Multi-Agent Systems. *Journal of Logic and Computation*, 8(3):401–424, 1998.
2. M. Esteva and C. Sierra. ISLANDER1.0 Language definition. Technical report, IIIA-CSIC, 2001.
3. J. Ferber and O. Gutknecht. A Meta-Model for the Analisys of Organizations in Multi-Agent Systems. In *Proc. 3rd Int'l Conf. on Multi-Agent Systems (ICMAS-98)*, pages 128–135, Paris, France, 1998.
4. P. A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1995.
5. The Foundation for Intelligent Physical Agents. See `http://www.fipa.org/`.
6. A. Ford. *Modeling the Environment: An Introduction to System Dynamics Models of Environmental Systems.* Island Press, USA, 1999.
7. The ISLANDER Web-Page. `http://www.iiia.csic.es/ISLANDER.html`.
8. N. R. Jennings. On Agent-Based Software Engineering. *Artificial Intelligence*, 117:277–296, 2000.
9. N. R. Jennings and M. Wooldridge. Agent-Oriented Software Engineering. In *Handbook of Agent Technology.* AAAI/MIT Press, 2000.
10. J. O. Kephart and A. R. Greenwald. Shopbot Economics. In S. Parsons and M. J. Wooldridge, editors, *Proc. 1st. Workshop on Game Theoretic and Decision Theoretic Agents*, pages 43–55, 1999.
11. J. O. Kephart and A. R. Greenwald. Probabilistic Pricebots. In P. Gymtrasiewicz M. J. Wooldridge, editor, *Proc. 3rd. Workshop on Game Theoretic and Decision Theoretic Agents*, pages 37–44, 2001.
12. J. A. Rodríguez Aguilar, F. J. Martín, P. Noriega, P. Garcia, and C. Sierra. *Towards a Formal Specification of Complex Social Structures in Multi-Agent Systems*, pages 284–300. Number 1624 in LNAI. Springer-Verlag: Berlin, Germany, 2000.
13. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

14. P. Noriega and C. Sierra. Towards Layered Dialogical Agents. In J. P. Müller, M. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III (LNAI Volume 1193)*, pages 173–188. Springer-Verlag: Berlin, Germany, 1997.
15. P. Noriega and C. Sierra. *Auctions and Multi-Agent Systems*, pages 153–175. Springer-Verlag: Berlin, Germany, 1999.
16. S. Parsons, C. Sierra, and N. R. Jennings. Agents that Reason and Negotiate by Arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.
17. SLIE Project. Web-Pages of Research Grant **Sustainable Lifecycles in Information Ecosystems** (SLIE). `http:// www.dai.ed.ac/groups/ssp/slie/`, 2000.
18. J. A. Rodríguez, F. J. Martin, P. Noriega, P. Garcia, and C. Sierra. Towards a Test-Bed for Trading Agents in Electronic Auction Markets. *AI Communications*, 11(1):5–19, 1998.
19. J. A. Rodríguez, P. Noriega, C. Sierra, and J. Padget. A Java-based Electronic Auction House. In *2nd Int'l Conf. on Practical Applic. of Intell. Agents & Multi-Agent Technology (PAAM'97)*, pages 207–224, London, UK, 1997.
20. C. Sierra, P. Faratin, and N. R. Jennings. A Service-Oriented Negotiation Model between Autonomous Agents. In *MAAMAW'97*, number 1237 in LNAI, pages 17–35, Ronneby, Sweden, 1997. Springer-Verlag: Berlin, Germany.
21. C. Sierra, N. R. Jennings, P. Noriega, and S. Parsons. A Framework for Argumentation-Based Negotiation. In M. P. Singh, A. Rao, and M. J. Wooldridge, editors, *Intelligent Agents IV (LNAI Volume 1365)*, pages 177–192. Springer-Verlag: Berlin, Germany, 1998.
22. H. Van Parunak, R. Savit, and R. L. Riolo. Agent-Based Modeling vs. Equation-Based Modeling: a Case Study and Users' Guide. In *Proc. Workshop on Modeling Agent-Based Systems (MABS98)*, Paris, France, 1998.
23. M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.
24. M. J. Wooldridge and S. D. Parsons. MABLE: An Agent Programming Language and its BDI Semantics. Unpublished manuscript.

A Interaction Specification: Remaining Cases

A customer in a negotiation can receive an offer of a deal from a supplier; then offers a contract to that supplier and moves to the delivery scene; or refuses the deal and moves to the agora as a trader:

```
agent(negotiation,customer,X) ::=
    offer(deal,_) <= agent(negotiation,supplier,Y) then
     ( ( offer(contract,_) => agent(negotiation,supplier,Y) then
         agent(delivery,customer,X)) or
       ( refuse(deal,_) => agent(negotiation,supplier,Y) then
         agent(agora,trader,X) ) ).
```

A supplier in a negotiation scene sends an offer of a deal to a customer; then receives a contract from that customer and moves to the delivery scene; or receives a refusal from the customer and moves to the agora as a trader:

```
agent(negotiation,supplier,X) ::=
    offer(deal,_) => agent(negotiation,customer,Y) then
    ( (  offer(contract,_) <= agent(negotiation,customer,Y) then
         agent(delivery,supplier,X) ) or
```

```
        (  refuse(deal,_) <= agent(negotiation,customer,Y) then
           agent(agora,trader,X) ) ).
```

A customer in a delivery scene receives notification of delivery or is informed of failure to deliver; then becomes a trader in the agora:

```
agent(delivery,customer,X) ::=
    ( offer(delivery,_) <= agent(delivery,supplier,Y) or
      inform(failure,_) <= agent(delivery,supplier,Y) ) then
    agent(agora,trader,X).
```

A supplier in a delivery scene sends notification of delivery or informs a customer of failure to deliver; then becomes a trader in the agora:

```
agent(delivery,supplier,X) ::=
    ( offer(delivery,_) => agent(delivery,customer,Y) or
      inform(failure,_) => agent(delivery,customer,Y) ) then
    agent(agora,trader,X).
```

This completes our specification of the potential interactions in the agent model.

B Decision Procedures: Remaining Cases

A broker that receives a message offering stock or requesting stock from a trader notes that it received the message:

```
agent(agora,broker,_) ::
  offer(stock,N) <= agent(agora,trader,X) -->
    assert(made_offer(X,N)).
agent(agora,broker,_) ::
  request(stock,N) <= agent(agora,trader,X) -->
    assert(made_request(X,N)).
```

A broker in the agora sends an offer of a buyer, X, to a trader, Y, if Y has made an offer to the broker and X has made a request to the broker and the amount requested is at least as much as that offered. The broker forgets about the original offers and requests and notes that it has chosen a seller. A broker will offer a seller, Y, to a trader, X, if it has a note that it chose Y as a seller to X:

```
agent(agora,broker,_) ::
  offer(buyer(X),No) => agent(agora,trader,Y) <--
    made_offer(Y,No) and made_request(X,Nr) and Nr >= No and
    retract(made_offer(Y,No)) and
    retract(made_request(X,Nr)) and assert(chose_seller(X,Y,Nr)).
agent(agora,broker,_) ::
  offer(seller(Y),Nr) => agent(agora,trader,X) <--
    chose_seller(X,Y,Nr) and retract(chose_seller(X,Y,Nr)).
```

A trader in the agora that receives an offer of a buyer notes that it has a potential customer. A trader in the agora which receives an offer of a seller notes that it has a potential supplier:

```
agent(agora,trader,_) ::
  offer(buyer(X),N) <= agent(agora,broker,_) -->
    assert(customer(X,N)).
agent(agora,trader,_) ::
  offer(seller(Y),N) <= agent(agora,broker,_) -->
    assert(supplier(Y,N)).
```

A customer in a negotiation scene can offer a contract to a supplier if the amount of stock it would have if it accepts the amount on offer from the supplier would top up its stock to no more than 50 units. Otherwise, if the amount offered would increase its stock to more than 50, it sends a message refusing the deal:

```
agent(negotiation,customer,_) ::
  offer(contract,N) => agent(negotiation,supplier,Y) <--
    received_offer(Y,N) and resource(stock,NS) and
    NT is N + NS and NT =< 50 and retract(received_offer(Y,N)) and
    retract(supplier(Y,_)) and assert(contract_to_buy(Y,N)).
agent(negotiation,customer,_) ::
      refuse(deal,N) => agent(negotiation,supplier,Y) <--
          received_offer(Y,N) and resource(stock,NS) and
          NT is N + NS and NT > 50 and
          retract(received_offer(Y,N)) and retract(supplier(Y,_)).
```

A customer in a negotiation scene that receives an offer of a deal from a supplier notes that it received an offer:

```
agent(negotiation,customer,_) ::
  offer(deal,N) <= agent(negotiation,supplier,Y) -->
    assert(received_offer(Y,N)).
```

A supplier in a negotiation scene can send an offer of a deal to a customer if it has a surplus of stock above 50 units. It offers the whole surplus:

```
agent(negotiation,supplier,_) ::
  offer(deal,N) => agent(negotiation,customer,Y) <--
    customer(Y,NC) and resource(stock,NS) and
    NL is max(NS - 50,0) and N is min(NL,NC) and N > 0.
```

A supplier in a negotiation scene that receives an offer of a contract notes that it has a contract to deliver and deletes its note of a potential customer. If it receives a refusal of a deal it simply deletes its note of a potential customer:

```
agent(negotiation,supplier,_) ::
  offer(contract,N) <= agent(negotiation,customer,Y) -->
    assert(contract_to_deliver(Y,N)) and retract(customer(Y,_)).
agent(negotiation,supplier,_) ::
  refuse(deal,_) <= agent(negotiation,customer,Y) -->
    retract(customer(Y,_)).
```

A supplier in a delivery scene can send an offer of delivery to a customer if it has a contract to deliver to that supplier and it has a surplus of stock above 50 units which it can commit. Otherwise, it informs the customer of its failure to deliver:

```
agent(delivery,supplier,_) ::
  offer(delivery,N) => agent(delivery,customer,Y) <--
    contract_to_deliver(Y,NC) and resource(stock,NS) and
    NL is max(NS - 50,0) and N is min(NL,NC) and N > 0 and
    retract(contract_to_deliver(Y,N)) and decrement(stock,N).
agent(delivery,supplier,_) ::
  inform(failure,N) => agent(delivery,customer,Y) <--
    contract_to_deliver(Y,NC) and resource(stock,NS) and
    NL is max(NS - 50,0) and N is min(NL,NC) and N =< 0 and
    retract(contract_to_deliver(Y,N)).
```

A customer in a delivery scene that receives an offer of delivery deletes its note of a contract to buy and increments its stock by the amount committed. If it receives a message informing it of failure it deletes its contract note:

```
agent(delivery,customer,_) ::
  offer(delivery,N) <= agent(delivery,supplier,Y) -->
    retract(contract_to_buy(Y,_)) and increment(stock,N).
agent(delivery,customer,_) ::
  inform(failure,_) <= agent(delivery,supplier,Y) -->
    retract(contract_to_buy(Y,_)).
```

C Formal Interactions in the 2-Agent Model

Given the specification above, we can simulate the interactions between agents for given sizes of rings. In the case of $\mathcal{R} = 2$, our interpreter obtains the term below, representing the sequence of message exchanges and changes in roles/scenes as the agents follow the electronic institution.

```
a(agent(agora,trader,a1),
  sent(request(stock,17)=>agent(agora,broker,b1)) then
  received(offer(seller(a2),17)<=agent(agora,broker,b1)) then
  a(agent(negotiation,customer,a1),
    received(offer(deal,17)<=agent(negotiation,supplier,a2)) then
    sent(offer(contract,17)=>agent(negotiation,supplier,a2)) then
    a(agent(delivery,customer,a1),
      received(offer(delivery,17)<=agent(delivery,supplier,a2)) then
      agent(agora,trader,a1)))),
a(agent(agora,trader,a2),
  sent(offer(stock,17)=>agent(agora,broker,b1)) then
  received(offer(buyer(a1),17)<=agent(agora,broker,b1)) then
  a(agent(negotiation,supplier,a2),
    sent(offer(deal,17)=>agent(negotiation,customer,a1)) then
    received(offer(contract,17)<=agent(negotiation,customer,a1)) then
    a(agent(delivery,supplier,a2),
      sent(offer(delivery,17)=>agent(delivery,customer,a1)) then
      a(agent(agora,trader,a2),
    (offer(stock,_)=>agent(agora,broker,A) then
     offer(buyer(_),_)<=agent(agora,broker,A) then
     agent(negotiation,supplier,a2))or
    request(stock,_)=>agent(agora,broker,B) then
```

```
    offer(seller(_),_)<=agent(agora,broker,B) then
    agent(negotiation,customer,a2))))),
a(agent(agora,broker,b1),
  (received(offer(stock,17)<=agent(agora,trader,a2)) par
   received(request(stock,17)<=agent(agora,trader,a1)) par
   sent(offer(buyer(a1),17)=>agent(agora,trader,a2)) then
   sent(offer(seller(a2),17)=>agent(agora,trader,a1))) then
   a(agent(agora,broker,b1),
     (offer(stock,_)<=agent(agora,trader,_) par
      request(stock,_)<=agent(agora,trader,_) par
      offer(buyer(_),_)=>agent(agora,trader,_) then
      offer(seller(_),_)=>agent(agora,trader,_)) then
      agent(agora,broker,b1)))
```