Mapping the Envelope of Social Simulation Trajectories

Oswaldo Terán[1], Bruce Edmonds and Steve Wallis
{o.teran, b.edmonds, s.wallis}@mmu.ac.uk
Centre for Policy Modelling
Manchester Metropolitan University
Aytoun Building, Aytoun Street, Manchester, M1 3GH, UK.
http://www.cpm.mmu.ac.uk/

Abstract

Discovering and studying emergent phenomena *are among the most important activities in social research. Replicating this phenomenon in "the lab" using simulation is an important tool for understanding it. Multi-Agent Systems (MAS) provide a suitable framework for such simulation. When such simulations are used to represent social processes there are necessarily indeterministic and arbitrary aspects, which are typically represented as either random choices (or numbers) or constants chosen by the programmer. Each such 'choice' means that the simulation takes one of the possible 'trajectories'. The implicit* theory *that a simulation represents is precisely* not *in the individual choices but rather in the 'envelope' of possible trajectories – what is important is the shape of the whole envelope. Typically a huge amount of computation is required when experimenting with factors bearing on the dynamics of a simulation to tease out what affects the shape of this envelope. In this paper we present a methodology aimed at systematically exploring this envelope. Thus it complements methods like Monte Carlo analysis, the inspection of single scenarios and syntactical proof. We propose a method for searching for tendencies and proving their* necessity *relative to a range of parameterisations of the model and agents' choices, and to the logic of the simulation language. The exploration consists of a forward chaining generation of the trajectories associated to such a range of parameterisations and agents' choices. Additionally, we propose a computational procedure that helps implement this exploration by translating the MAS simulation into a constraint-based search over possible trajectories by 'compiling' the simulation rules into a more specific form, namely by partitioning the simulation rules using appropriate modularity in the simulation. An example of this procedure is exhibited.*

KEYWORDS: Social Simulation, Multi-agent Systems, Model, Proof, Emergence, Tendencies

1. Social Simulation and the Exploration of Simulation Trajectories

A social simulation necessarily abstracts from some idea about processes that produce social phenomena. Typically this means that: *firstly*, many of the simulation parameters will be in essence chosen arbitrarily and, *secondly*, that there will be indeterministic choice processes in the simulation to 'stand in' for processes which we do not want to simulate. In particular a pseudo-random number generator often 'stands in' for some aspect of a real choice made by a social actor or some unpredictable aspect of the target environment. One can think of each choice as resulting in a branch point in the simulation – where the simulation trajectory 'branches out' into a separate trajectory for each possibility. The intended content of the simulation is exactly not the individual trajectories, but the envelope of these trajectories.

It may be that every branch diverges from the others so that the result is completely *contingent* upon the exact choices made. On the other hand it may be that all branches share a common tendency or converge to the others in certain aspects. This commonality could be explicitly 'forced' by the design of the simulation in the form of an explicit constraint: for example if a room has only one exit then the actors in that room may all exit by the same door eventually *whatever* the nature of their individual choice processes. In other simulations the commonality is *emergent* in the sense that it is difficult to explain the commonality between possibilities in terms of the simulation design – this is what we call emergent

[1] Also a member of the Centro de Simulación y Modelos (CESIMO: Centre for Simulation and Modelling) and the Department of Operations Research of the University of Los Andes, Venezuela (http://cesimo.ing.ula.ve/).

tendencies. This paper documents some steps in the search for ways to understand such emergent tendencies.

A simulation study can have many purposes, including these: it may help in the understanding of some phenomena and also it may inform the design of future simulations. Exploring possible simulation trajectories and analysing the resulting dynamics of the simulation are central to both these tasks. Usually there is a trade-off between the richness of the study in terms of the number of explored trajectories (sometimes related to how fine-grained the model is) and the amount of required computational resources. The finer the model the more "realistic" the simulation model will be, but also the more intricate the analysis of the simulation will be.

A typical case where this analysis is crucial is in Multi-Agent Based Social Simulation. There, modellers may attempt to generate in the lab certain "complex" behaviours in a whole population as the result of the interaction of simpler. Unforeseen behaviour of individuals and unpredictable tendencies in the behaviour of the whole population can arise [4].

The lack of alternative methodologies and tools for appropriate exploration and analysis of the dynamics of a simulation are presently a factor, which limits the comprehension of emergent tendencies. Present methods include examining individual trajectories as in Scenario Analysis [3] and statistical sampling as in Monte Carlo techniques [12]. Each of these has its limitations. It is our purpose in this paper to complement these with an alternative way of exploring and analysing the simulation by systematically and automatically mapping the envelope of *all* possible trajectories in a substantial fragment of a simulation.

2. Enveloping Tendencies in Simulation Trajectories: a Constrained Search over Possible Models

The traditional methods for examining simulation trajectories are: Scenario Analysis and Monte Carlo techniques.

Via *Scenario Analysis* trajectories are inspected one at a time and as many alternatives as possible examined. Nevertheless, it is usually unviable to map all the possibilities, as the number of alternative trajectories is far too large. Additionally, the high amount of data increases the difficult task of searching for exceptional behaviour displayed by (groups of) agents in a simulation. Moreover, it is left up to the modeller to make conclusions about the persistence and sensitivity of certain behavioural outcomes with respect to a certain range of the factors.

On the other hand, a *Monte Carlo analysis* explores the dynamics of the simulation via statistical analysis of quantitative change (or quantitative measures of qualitative changes) observed in a sample of trajectories. The sample is done over the range of possibilities given by random variables introduced in the model to simplify uncertainties. The difficulties with this are that: it tells us what is a probable outcome rather than what is a necessary outcome and it can involve the use of inappropriate statistical assumptions.

2.1 Constrained exploration of trajectories

We propose the use of an exhaustive constraint-based search over a range of possible trajectories in order to establish the necessity of postulated emergent tendencies. Thus a subset of the possible simulation parameterisations and agent choices are specified; the target emergent tendencies are specified in the form of negative constraints; and an automatic search over the possible trajectories performed. The tendencies are shown to be necessary with respect to the range of parameterisations and indeterministic choices by first finding a possible trajectory without the negative constraint to show the rules are consistent and then showing that all possible trajectories violate the negation of the hypothetical tendency when this is added as a further constraint. (See figure 1).

2.2 Characterising the envelope of tendencies

In order to distinguish between the *exceptional* and the *representative* in a simulation, we will formally describe the envelope of certain tendencies in a simulation. This might be done by:

- Certain *properties* satisfied by the observed tendency.
- A *mathematical description* of a subspace of the tendencies or of a subspace given a bound of the tendencies.
- *Representative or typical instances* of such a tendency.
- A *mapping* from the setting of trajectories, as given by the alternative arrangement of parameters and agents' choices, to certain knowledge (maybe properties) about the tendency: *(parameters* X *choices)*→ *(know. of the tend.)*

2.3 Proving the necessity of a tendency

We want to be able to *generalise* about tendencies going from observation of individual trajectories to observation of a group of trajectories generated for certain parameters and choices. Actually, we want to know if a particular tendency is a necessary consequence of the system or a contingent one. For doing this we propose to *translate* the original MAS along with the range of parameterisations and agents' choices into a platform (described in the next section) where the alternative trajectories can be *unfolded*. Each trajectory will correspond to a possible trajectory in the original MAS.

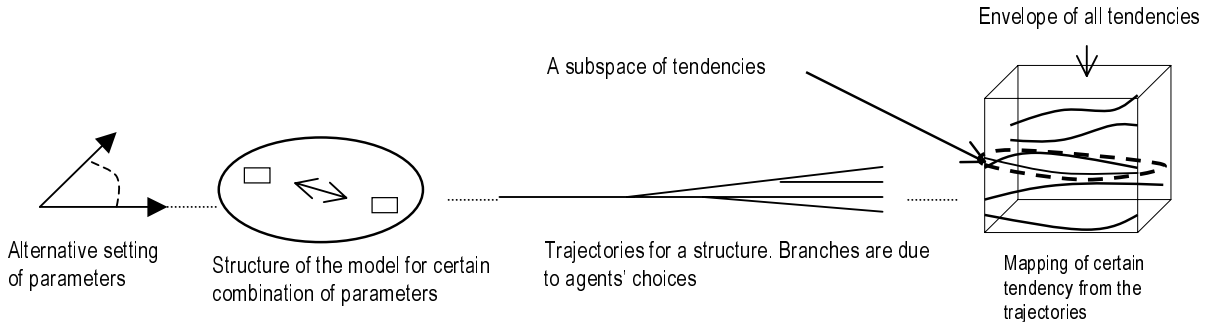

Figure 1. A constraint-based exploration of possible simulation dynamics

Once one trajectory is shown to satisfy the postulated tendency another set of parameters and agents' choices is selected and the new trajectory is similarly checked. If all possible trajectories are successfully *tested*, the tendency is *proved to be necessary* relative to the logic of the simulation language, the range of parameterisations and agents' choices.

The idea is to translate the MAS into a constraint-based platform in an automatic or near automatic way without changing the *meaning* of the rules that make it up in order to perform this automatic testing. In this way a user can program the system using the agent-based paradigm with all its advantages; inspect single runs of the system to gain an intuitive understanding of the system and then check the generality of this understanding for fragments of the system via this translation into a constraint-based architecture.

In the *example* shown below, all trajectories are explored for one combination of parameters, eight agents' choices per iteration and seven iterations. A simple tendency was observed characterised by a mathematical description of its boundaries. This characterisation was handled as a theorem. The theorem was proved to be necessary following a procedure similar to the one described in the previous paragraph.

2.4 What is new in this model-constrained methodological approach

It is our goal in this paper to propose an alternative approach for exploring and analysing simulation trajectories. It will allow the entire exploration and subsequent analysis of a subspace of the whole space of simulation trajectories. We are suggesting the generation of trajectories in a semantically constrained way. Constrictions will be context-dependent (over the semantics of the trajectory itself) and will be driven via the introduction of a controller or meta-module.

Like Scenario Analysis, the idea is to generate individual trajectories for different parameterisations and agents' choices but unlike Scenario Analysis the exploration is constrained to only certain range of parameters and choices.

Akin to Monte Carlo techniques it explores only part of the total range of possible trajectories. But, unlike Monte Carlo studies it explores an entire subspace of (rather than some randomly generated sample) trajectories and is able to give *definitive* answers for inquires related to the dynamics of the simulation in that subspace.

3. Towards the implementation of a suitable platform for the envelope of trajectories: using SDML and Declarative Programming Paradigm

SDML (Strictly Declarative Modelling Language) [8] is the declarative Multi-Agent System in which we have developed our experiments. As a source of comparisons and ideas, we have also programmed our model in a Theorem Prover [2,7,10,11]

In general, declarative programming (and in particular SDML) offers desirable features for simulation experiments as compared to imperative programming. For the social simulation community those features seem to be of particular interest when facilitating the exploring and analysis of the dynamics of the simulation [9].

3.1 Some characteristics of Declarative Programming

- *Modularity*. Any part of the model is constructed as a group of standardised units (i.e. rules) allowing *flexibility* and *variety* in use. The declarative paradigm facilitates a greater level of modularity than the imperative paradigm because the *control* of the program is separated from the *content*. This flexibility is useful both when representing the static structure of the system and when generating the dynamics of the simulation. In our case, it facilitates the introduction of alternatives for agents' choices and parameters of the model.
- *Expressiveness*. Effective conveyance of *meaning* is a consequence of the representation of the system as

linguistic clauses on a set of databases. It facilitates the interpretation of a set of social phenomena into a simulation by allowing the dual interpretation of clauses as pseudo-linguistic tokens and as entities to be computationally manipulated.

- *Easier analysis*. Context situated *analysis* of detailed data, tracks of trajectories as well as analysis of group of trajectories is much more straightforward than in imperative programs because the resulting databases can be flexibly browsed and queried.
- *The Possibility of Formal Proof.* The data generated by the dynamics of the simulation can be analysed as a *logical extension* under the particular logic of the simulation language. It opens the possibility of achieving *proofs* related to the logic of the language and the constraints imposed by the allowed choices of the agents and parameters of the model.

3.2 Other relevant characteristics and features SDML offers

- Good underlying logical properties of the system. SDML's underlying logic is close to the Strongly Grounded Autoepistemic Logic (SGAL) described by Kurt Konolige [6].
- Its backtracking procedure facilitates the exploration of alternative trajectories via the splitting of simulation paths according to agent's choices and model's parameters.
- The assumptions manager in SDML tracks the use of assumptions. Assumptions result from choices.
- A collection of useful primitives relevant to social simulation.
- The type meta-agent. A meta-agent (meta, for our purposes) is an agent "attached" to another agent as a controller; it is able to program that agent. This is used here *not* as an agent *per se* but as a module used to 'compile' rules into an efficient form as well as to monitor and control the overall search process and goals.

4. Implementing a suitable constraint-based programming platform.

The main goal of the programming strategy to be described is to increase the efficiency in terms of simulation time, thus making an efficient constraint-based search possible. The improvements will be achieved by making the rules and states more context-specific. This enables the language's inference engine to exploit more information about the logical dependencies between rules and thus increase the efficiency. Thus this can be seen as a sort of 'practical compilation' process, which *undoes* the agent encapsulation in order to allow the more efficient exploration of its behaviour. In particular we split the transition rules into one per simulation period, and also by the initial parameters. This necessitates a dynamic way of building rules. This is done via a controller, which generates the rules at the beginning of the simulation.

4.1 An Overview of the system.

We implemented the proposed architecture in three modules; let us call them **model**, **prover** and **meta**. The following diagram illustrates this:

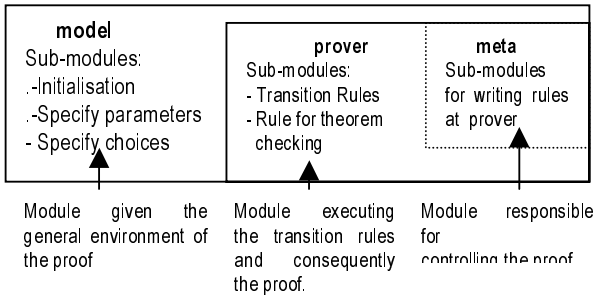

Figure 2. Physical view of the system.

4.2 Description of System Modules

We have found it convenient to *distinguish* and model as distinct entities three basic elements of a simulation: the *static structure* of the model, the *dynamics* of the simulation and the way this dynamics is "managed" by certain meta-rules or by a *controller*. Each of those entities is programmed in a different module:

1. **model**, *sets up the structure* of the model, that is, it gives the environment of the simulation: range of parameters, initialisations, alternative choices and basic (backward chaining) rules for calculations.
2. **prover**, *generates the dynamics* of the simulation. This is a sub-module of *model* (i.e. it is contained in *model*). This will basically contain the transition rules, auxiliary rules for generating pre-processing required data and the conditions to test the necessity of the theorem. All of them are rules to be executed while the simulation is going on.
3. **meta**, *is responsible for controlling the dynamics* of the simulation. Its meta-rules write the transition rules and the theorem in (as well as others required by) the module *prover*. A picture of the system is given in *Figure 2.*

4.3 Program dynamics

Modules' rules are executed in the following *sequence:*

1. **model**: initialising the environment for the proof (setting parameters, etc..)
2. **meta**: creating and placing the transition rules in *prover*.

3. **prover**: carrying on the simulation using the transition rules and backtracking while a contradiction is not found.

The program *backtracks* from a path once the conditions for the theorem are verified, then a new path with different choices and/or parameters is picked up.

4.4 Split of the rules: a source of efficiency.

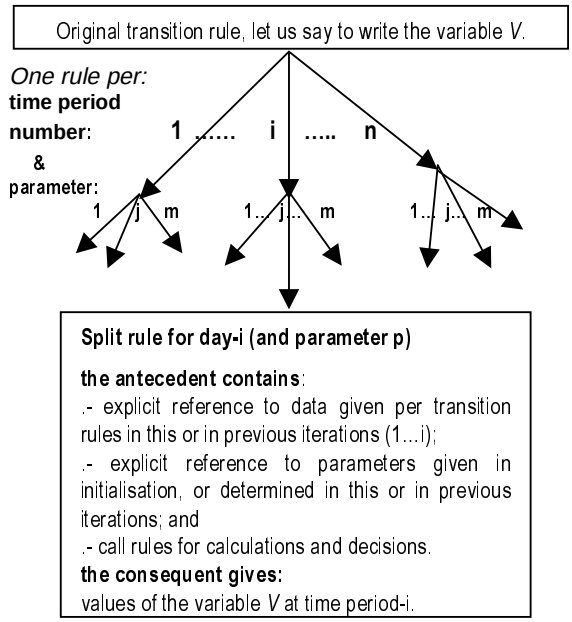

Figure 3. Splitting of rules by time period and a combination of parameters.

In forward chaining simulation the antecedent retrieves instance data from the past in order to generate data for the present (and maybe the future):

past facts → present and future facts

Traditionally, the set of transition rules are implemented to be general for the whole simulation. A unique set of transition rules is used at any STI (Simulation Time Instant or iteration).

As the simulation evolves, the size of the database increases and the antecedents have to discriminate among a growing amount of data. At STI_i, there would be data from *(i-1)* alternative days matching the antecedent. As the simulation evolves it becomes slower because of the discrimination the program has to carry out among this (linearly) growing amount of data.

Using the proposed technique, we would write a transition rule for each simulation time. The specific data in the antecedent as well as in the consequent could be instanced. Where possible, a rule for each datum, the original rule will generate, would be written. This will be illustrated in the example of the next section.

This technique represents a step forward in improving the efficiency of declarative programs. One could, in addition, make use of partitions and time levels to introduce further modularity – this would further speed up the search process and increase the memory that is needed. Partitions permit the system to order the rules to fire in an efficient way according to their dependencies. Time levels let us discriminate among data lasting different amounts of time. The *splitting of rules lets us discriminate among the transition rules for different simulation times given a more specific instancing of data at any STI.*

4.5 Measuring the efficiency of the technique

Comparing the two programs, the original MAS simulation and the constraint-based translation we obtain a *speed up* by a factor of $O(NM)$, where N is the average number of agents instantiated by a rule and M is the number of STIs. SDML already has facilities for discriminating among STIs, but their use is not convenient for the sort of simulation we are doing (exploring scenarios and/or proving) because of the difficulties for accessing data from any time step at any time. If we had used this facility still the simulation would have been speeded up by N. Notice that all these values are only estimations because a program stops trying to fire a rule as soon as it finds out that one of its clauses is false.

It is clear that the greater the number of entities in the simulation or the number of STIs, the larger the benefits from the technique. We must notice that the speeding up of the simulation is only one dimension of the efficiency given by the technique.

5. An example.

A simple model of a producer-consumer system, which was previously built in SDML and in the Theorem Prover OTTER, was rebuilt using the proposed modelling strategy. In the new model the exploration of possibilities is speeded up by a factor of 14. Also, the model built in OTTER, though faster than the original model in SDML, is several times slower than the improved model built in SDML.

Some of the split transition rules were the ones for creating (at each STI) producers' prices and sales, consumers' demand and orders, warehouses' level and factories' production. Among the rules for auxiliary data split were the ones for calculating: total-order and total-sales (a sum of the orders for all producers), total-order and total-sales per producer, and total-order and total-sales per consumer.

5.1 Example of a split rule: *Rule for prices*.

This rule calculates a new price for each producer at each STI (which we called *day*), according to its own

price and sales, and the price and sales of a chosen producer, at the immediately previous STI.

The original rule in SDML was like this:

```
for all (producer)
for all (consumer)
for all (day)
(
 price(producer,myprice,day)                 and
 totalSales(totalSales,day)                  and
 sales(producer,mySales,day)                 and
 choiceAnotherProducer(anotherProducer)      and
 price(anotherProducer,otherPrice, day)      and
 calculateNewPrice(mySales,totalSales,
 otherPrice, myPrice,newPrice)
implies
 price(producer, newPrice, day + 1)
   )
```

The new rule (in the efficient program) will be "broken" making explicit the values of prices and sales per each day.

In the following, we show the rule per *day-i* and *producer-j*:

```
for all (consumer)
(
 price(producer-j, myprice, day-i)           and
 totalSales(totalSales, day-i)               and
 sales(producer, mySales, day-i)             and
 choiceAnotherProducer(anotherProducer)      and
 price(anotherProducer, otherPrice, day-i)   and
 calculateNewPrice(mySales,totalSales,
 otherPrice, myPrice,newPrice)
implies
 price(producer-j, newPrice, (day-i) + 1)
)
```

If the name of price is used to make explicit the day, the rule will have the following form. It is important to observe that *only one instance of newprice in the consequent is associated with only one transition rule and vice verse*:

```
for all (consumer)
(
 price-i(producer-j, myPrice)                and
 totalSales-i(totalSales)                    and
 sales-i(producer-j, mySales)                and
 choiceAnotherProducer(anotherProducer)      and
 price-i(anotherProducer, otherPrice)        and
 calculateNewPrice(mySales,totalSales,
 otherPrice, myPrice,newPrice)
implies
 price-(i+1)(producer-j, newPrice)
```

5.2 What the technique enables

In this example, the described technique was used to prove that the size of the interval of prices (that is*: biggest price - smaller price*, each day) decreases over time during the first six STIs over a range of one parameterisation and eight choices for the agents at each STI. An exponential decrease of this interval was demonstrated in all the simulation paths. A total of 32768 simulation trajectories were tested. It was not possible to simulate beyond this number of days because of the limitations imposed by computer memory. The complete search process took only 24 hours.

Though the tendency we have shown is simple and quantitative, it is obvious that the technique is applicable in more interesting cases of emergent tendencies, even if they have a qualitative nature.

This technique is useful not only because of the speeding up of the simulation but also for its appropriateness when capturing and proving tendencies under the specified constraints. In the example, the meta-module was used to write the rule with the hypothesis (theorem) to be tested on prover-module at the beginning of the simulation. If the meta-module were able to write rules on prover-module while the simulation is going on, the theorem we wanted to prove could be *adapted* according to the results of the simulation via *relaxing constraints*. For example, the technique could be implemented in a way that we only give the program hints related to the sort of proof we are interested in. Then the meta-module would *"elaborate"*, via adapting over time in a context dependent manner, a set of hypothesis or theorems. That is, a *theorem into a family of alternative theorems will be searched* (those hints will specify such a family) starting from the most constrained one.

6. Other Approaches

6.1 Using OTTER [7], a resolution-style first order Theorem Prover.

In simulation strategies like event-driven simulation or partition of the space of rules, in declarative simulation, are used. The criteria for firing rules is well understood, and procedures like weighting and subsumption usually are not necessary. Additionally, redundant data for some purpose could be avoided in MAS with appropriate compilation techniques.

The advantages given for the weighting procedure in OTTER are yielded in MAS systems like SDML by procedures such as *partitioning*, where chaining of the rules allows firing the rules in an efficient order according to their dependencies.

6.2 DESIRE

Among other approaches for the practical proof of MAS properties, the more pertinent might be the case conducted by people working in DESIRE [5]. They propose the hierarchical verification of MAS properties, and succeeded in doing this for a system.

However, their aim is the verification of a computational program – it is proved that the program behaves in the intended way. It does not include the more difficult task, which we try to address, of establishing general facts about the dynamics of a system when run or

comparing them to the behaviour observed in other systems [1].

7. Conclusions and future work

We have argued and shown using an example the pertinence of a methodology for a constrained exploration and envelope of trajectories as complement to traditional methods dealing with post-hoc analysis of the dynamics of simulations. We have suggested a forward chaining generation of trajectories in a semantically constrained way. Constrictions will be context-dependent (over the semantic of the trajectory itself) and will be driven via the introduction of a controller. Once a tendency is identified the idea is to prove its necessity relative to the logic of the simulation language, a range of parameterisations and agents' choices.

A platform to implement this methodology has been proposed. It consists of a modular structure according to strategic parts of a simulation: a first module, *model*, sets up the *static structure* of the simulation; then a second module, *prover*, generates the *dynamics* of the simulation; and finally a *meta-module* is responsible for *controlling* the dynamics of the simulation. The second characteristic of this platform is a *partitioning* of the space of rules and *splitting of transition rules* by STI, parameters and choices.

This technique represents a step forward in improving the efficiency of declarative programs, one additional to the use of partition and time levels. The *splitting* of rules let us to *discriminate* among the transition rules for different simulation times steps given a more *specific* instancing of data at any STI. Thus this alleviates some of the drawbacks of declarative programming due to the necessary grasping and updating of all state variables at any STI.

In the example the meta-module acts only at the beginning of the simulation. More powerful and flexible programs are possible when the meta-module acts at any STI. Transition rules can be evolving, in a sense that the meta-module builds the rules for a STI once every fact for the previous STI are known. This allows adaptation of the rules (e.g. relaxing in the constraints defining the theorem) according to the changing environment of the exploration.

Our future work will be focused both in the search for a refining and improving of the presented technique and in applying it to more interesting cases of emergent tendencies for the social simulation community.

Acknowledgements

SDML has been developed in VisualWorks 2.5.2, the Smalltalk-80 environment produced by ObjectShare. Free distribution of SDML for use in academic research is made possible by the sponsorship of ObjectShare (UK) Ltd. The research reported here was funded by CONICIT (the Venezuelan Governmental Organisation for promoting Science), by the University of Los Andes, and by the Faculty of Management and Business, Manchester Metropolitan University.

References

[1] Axtell, R., R. Axelrod, J. M. Epstein, and M. D. Cohen, "Aligning Simulation Models: A Case of Study and Results", Computational Mathematical Organization Theory, 1(2) (1996), pp. 123-141.

[2] Chiang, C.L., and R. C.T. Lee, Symbolic Logic and Mechanical Theorem Proving, Academic Press, London, UK, 1973

[3] Domingo C., G. Tonella and O. Terán, "Generating Scenarios by Structural Simulation", in AI, Simulation and Planning High Autonomy Systems , The Univ. of Arizona (1996) , pp 331-336.

[4] Edmonds, B. , "Modelling Bounded Rationality In Agent-Based Simulations using the Evolution of Mental Models". In Brenner, T. (Ed.), Computational Techniques for Modelling Learning in Economics, Kluwer (1999), 305-332.

[5] Engelfriet, J., C. Jonker and J. Treur, "Compositional Verification of Multi-Agent Systems in Temporal Multi-Epistemic Logic", AI Group, Vrije Universiteit Amsterdam, The Netherlands, (1998).

[6] Konolige, K. , "Autoepistemic Logic", in Handbook of Logic in Artificial Intelligence and Logic Programming (4),Oxford Science Publications, 1995, pp. 217-295.

[7] McCune, W. (1995), OTTER 3.0 Reference Manual Guide, Argonne National Laboratory, Argonne, IL, 1995.

[8] Moss, S., H. Gaylard, S. Wallis, B. Edmonds, "SDML: A Multi-Agent Language for Organizational Modelling", Computational Mathematical Organization Theory, 4(1) (1998), 43-69.

[9] Moss, S., B. Edmonds, S. Wallis, "Validation and Verification of Computational Models with Multiple Cognitive Agents", Centre for Policy Modelling Report (1997), CPM-97-25, `http://www.cpm.mmu.ac.uk/cpmrep25.html`

[10] Wos, L., Automated Reasoning: 33 Basis Research Problems, Prentice Hall, New Jersey, USA, 1988.

[11] Wos, L., Robinson, G. A., and Carson, D. F., "Efficiency and Completeness of the Set of Support Strategy in Theorem Proving", J. ACM 14, N° 4 (1965), 698-709.

[12] Zeigler, B., Theory of Modelling and Simulation, Robert E. Krieger Publishing Company, Malabar, Fl, USA, 1976.