Comparing simulation models in a concurrent environment: the case of innovation diffusion

Jorge Louçã
LabMAg – Laboratory of Agent Modelling
ISCTE cacifo 296,
Av.das Forças Armadas
1649-026 Lisboa  Portugal
Jorge.L@iscte.pt

Valmir Meneses
valmir.meneses@gmail.com

Abstract
This paper proposes a generic software architecture for the concurrent execution of social simulation models. Concurrent execution of simulation models has considerable advantages regarding other ways of comparing models, such as run-time comparison, performance evaluation, and the possibility of interaction between complementary models. A tool has been implemented and experiments have been carried out comparing two different models, based on cellular automata and social network technologies, both applied to the domain of innovation diffusion.

Keywords: concurrent multiagent-based simulation, model-to-model comparison
1. Introduction

The scientific domain of social simulation has evolved, from its first experiences of designing ad hoc simulations, to the actual phase of categorizing simulation models. Examples of these categories are cellular automata (Wolfram 2002) or multiagent-based simulation models (Sallach 2006). Within these large categories we can identify more specific types of models, according to environment and agents’ parameters, or according to the way of representing social interactions.

The actual stage of the social simulation domain is characterized by the development of methodologies and tools allowing knowledge accumulation within its scientific community. Three challenges have been referred to in literature: comparing and reusing models, aiming to develop a comparative methodology, including the identification of applicable contexts and the limits of applicability (Cioffi-Revilla 2003); validating results obtained from a given model through the application of alternative models (Axelrod 1997); and interaction between models, looking for new solutions (Rouchier 2003).

This paper contributes to the development of a new generation of tools to support model comparison, interaction and validation. A generic architecture allowing concurrent execution of models is here proposed, where simulation models are compared through their application to a common situation. Our point of view is that concurrent execution of simulation models, where these share computational resources at run-time, has important advantages regarding classical forms of comparing models. Examples of those advantages are performance evaluation, comparison at run-time and the possibility of interaction between complementary models. This approach has been implemented in a tool that enables concurrent simulations. Experiments have been carried out on the comparison of two different models, respectively based on cellular automata and social network technologies, both applied to the domain of innovation diffusion (Edwards 2003).

This work integrates a larger research project, started in September 2006. The DIVERSITY project aims to propose modeling and programming tools to set up social simulations in computer networks. Theoretical and practical propositions are provided to the community, including (1) a general methodology for the formalization and design of concurrent and distributed social simulation models, allowing the development of a typology of MABS and (2) a programming library to implement concurrent and distributed simulations (Diversity 2006). This research is the natural sequence to the previous work of Valmir Meneses and Jorge Louçã, where a concurrent extension to REPAST was proposed (Meneses 2006) and new tools were proposed to study innovation processes (Louçã 2006).

The main body of the paper is divided into four sections. First the advantages of a concurrency-based architecture in the social simulation domain and a brief state-of-the-art of concurrency in MABS are described, followed by the proposition of a generic concurrent architecture for model comparison. Next, the implementation of this architecture is presented, comparing the use of two models based on cellular automata and social network technologies, and both are applied to the domain of innovation
diffusion. Several experiences using this implementation are described. The text ends with a view of planned extensions of this research.

2. State of the Art

2.1. Advantages of concurrency in distributed simulation architectures

From the technical point of view, the existing social simulation programming libraries are oriented towards the implementation of toy simulations, where agents and interaction rules tend to be very simple. Simulation models tend toward simplicity due, on the one hand, to a shortage of computational resources to manage quantities of interactions between complex artificial agents, and on the other hand to its incapacity for making good quality models of parallel real world entities. A way of improving the performance of simulation models, as well as extending their modeling abilities to the real world parallelism, is to conceive concurrent simulation models (Wellings 2004) (Diversity 2006).

Concurrent programs are characterized by the existence of multiple light processes, allowing the execution of different concurrent computations as if they were in parallel (Andrews & Schneider 1983) (Magee & Kramer 1999). The advantages of adopting concurrency, in the context of social simulation, are the following (Silberschatz et al. 2002):

- Improving the use of processors: in multi-processing environments the advantage is clear – each processor can attend a different process in parallel, improving the performance of the simulation and allowing interaction between sophisticated agents. On the other hand, in mono-processor environments the use of light processes optimizes the division of the available computational resources through the concurrent management allowed by the operating system.

- Modeling the parallelism of the real world: concurrent models have the ability to represent the distributed nature of social systems. Moreover, concurrency introduces event-ordering uncertainty because concurrently running agents execute at arbitrary rates relative to each (Gasser 2005). This allows designing complex simulation environments with important great degrees of freedom in event ordering due to concurrent execution.

2.2. Existing solutions

According to Wooldridge (1998), an obvious difficulty observed in many multiagent tools is the absence or poor use of concurrency. Concurrency is one of the more important potential qualities of multiagent systems, but it is frequently unexplored in this kind of system. More recently, Bitting et al. (2003) identified concurrency as an important issue in evaluating multiagent development packages. Duvigneau et al. (2003) verified that the majority of multiagent frameworks have a restrict support to concurrent
development, taking existing alternatives away from essential concurrency notions, such as synchronization and coordination for resource sharing. Nevertheless, some recent propositions have been made, adding concurrency to multiagent systems. For example, MACE3J (Gasser et al. 2005) simulates MAS models across a variety of scales and architecture types, from single PCs, to Single System Image (SSI) multicomputers, to heterogeneous distributed Grid environments, including different sorts of concurrent environments. Brahms (Sierhuis et al. in press) is a Java multiagent language used to simulate distributed work processes, having the capability of running several virtual machines. Agents register with a directory service and communicate through a Corba layer, representing distribution over a network. However, one of the known consequences of using Corba-based technologies is attaining poor overall performances in distributed systems.

Specifically concerning multiagent-based simulation tools, concurrency is not yet addressed as a fundamental issue: Swarm (Terna 1998) has some sort of concurrency allowing the instantiation and scheduling of activities as light processes, but this library does not permit the existence of autonomous concurrent agents; Repast (North et al. 2006) has some sort of simulated concurrency through an implementation of its notion of discrete event scheduler, not allowing the direct implementation of really concurrent processes; Ascape (Parker 2001) does not implement concurrent activities; Mason (Luke et al. 2004) is a multiagent simulator following an architecture similar to Swarm, and therefore does not allow agent concurrency; Madkit (Gutknecht 2000) is a versatile tool, allowing the development of concurrent agents using its basic classes. However, the synchronous scheduler of Madkit behaves like a discrete event simulator and for that reason the result is some kind of concurrency restricted by the scheduler. Finally, Tagus (Meneses 2006) is an extension to the Repast library, allowing the implementation of each agent as a light process. This extension permits the real concurrent execution of the agent’s actions, as well as the use of synchronization primitives to resource sharing between the concurrent agents. These characteristics make the complementarities between the tools Repast and Tagus a good solution to the concurrent representation of social simulations, and this was our technical choice for implementing the following concurrent architecture.

3. A generic architecture for the concurrent execution of social simulation models

3.1. Models comparison

This paper proposes a generic software architecture for the concurrent execution of social simulations, allowing the comparison of different simulation models. The description of this architecture starts with the clear statement of the notions of “simulation model” and “simulation execution” that will be used in the paper. A simulation model is identified, on the one hand, by the characterization of a social environment and, on the other hand, by the characterization of one or more classes of agent that interact within this environment. A simulation model corresponds to the scheme of a given social situation. The execution
of a simulation corresponds to the instantiation of the variables characterizing a model and by the execution of a certain number of interactions between its agents.

The fundamental idea underlining the architecture concerns the composition of a “simulation of simulations”, where different simulation models evolve within a basic support simulation environment. The execution of different simulations is the result of the instantiation of different models using the same parameters, taken from a given case study. This idea is informally illustrated in the following figures. The particular operation of model instantiation is formally represented using the Z notation (Spivey 2006).

The simulation environment is characterized by a set of attributes and a scheduler, with the following functionalities:

- **Attributes**
  - Case study parameters – the same case study must be applied to different comparable models, in order to guarantee coherence and compatibility of results. The characteristics of a common case study are identified in the simulation environment and are shared by the simulation models.
  - Models base – identifies the structural characteristics of different simulation models.

- **Comparative simulation scheduler** – the simulation environment scheduler is responsible for launching the execution of concurrent simulations. This scheduler chooses which models to compare and parameterizes them with the specific

![Generic architecture for the concurrent execution of simulation models](image)

Figure 1. Generic architecture for the concurrent execution of simulation models
characteristics of a given case study. Then it launches light processes, each one responsible for the execution of a simulation. This way, models are executed in a concurrent manner in the operating system.

The primary operation, prior to launching the execution of concurrent simulations, is the operation of \textit{model instantiation}, depicted in Figure 1 using dot and dash arrows. This operation is responsible for creating instances of models from models structures and from some case study set of parameters. A representation of the operation of \textit{model instantiation} is a central element of some general representation of the architecture. The formal specification language Z (Spivey 2006) is used to formally characterize this operation:

\begin{equation*}
\text{ModelInstantiation} = \Delta (\text{Models}, \text{CaseStudyParameters}, \text{InstantiatedBy})
\end{equation*}

\begin{equation*}
\text{Model?} : X
\end{equation*}

\begin{equation*}
\text{CaseStudyParameters?} : Y
\end{equation*}

\begin{equation*}
\text{Model?} \in \text{Models}
\end{equation*}

\begin{equation*}
\text{CaseStudyParameters?} \in \text{CaseStudyParameters}
\end{equation*}

\begin{equation*}
\forall x : X \cdot \exists y : Y \mid
\begin{align*}
x \mapsto y \\
\text{Model!} = \text{Model?} \cup \{ x \mapsto y \}
\end{align*}
\end{equation*}

Figure 2. Formal specification of the \textit{ModelInstantiation} operation

The \(\Delta\) operation alerts us to the fact that the schema corresponding to the \textit{ModelInstantiation} operation describes a state change. \textit{InstantiatedBy} is the set of tuples with first element a variable belonging to \textit{Model}? and second element a value in the set \textit{CaseStudyParameters}. \textit{Model}? is an input to the operation, representing the structure of a model before instantiation. \textit{CaseStudyParameters}? is also an input to the operation, composed by a set of values corresponding to the variables of \textit{Model}?. The output of the operation is \textit{Model!}, the instantiated model. After the \textit{ModelInstantiation} operation, the instantiated models are embedded in the simulation environment, allowing the concurrent execution and run-time comparison of a variety of simulation models (Figure 3).
The realization of real-time comparisons in such a concurrent context allows, on the one hand, the quantitative evaluation of performances concerning the use of computational resources by each model and, on the other hand, the qualitative evaluation of the results obtained by each model (Figure 3). The analysis of these results allows the verification of the adaptability of each model to the case study. Also, the validity of the results of a model is reinforced when confirmed by the results of other models.

### 3.2. Models synchronization

The main original advantage of this architecture above stands on the possibility of runtime interaction between models being executed concurrently. Actually, concurrency management mechanisms provided by operating systems permit the synchronization of simulations using pre-determined synchronization points (Figure 4).
The synchronization of models is a feature that allows us to idealize a new generation of simulation models, hybrid by nature, resulting from the cooperative execution of complementary models. Two aspects strongly concern this new generation of concurrent simulation models: resource sharing and coordination. The first is related to the share of input and intermediary data. To do so, synchronization primitives must be respected. On the other hand, the coordination of concurrent simulations should be accomplished in the cooperative simulation scheduler, allowing the definition of synchronization points, exchanging data and, generally, implementing interactions between simulations.

The next section presents some experimental results of our approach, concerning comparing results and performances. Other experiments, concerning the synchronization of complementary models, are currently being carried out in the laboratory, in the context of the DIVERSITY project (2006).

4. Experimental results concerning concurrent models: the case of innovation diffusion

4.1. First example – the case of innovation diffusion

The experimentation of our propositions was carried out by comparing two simulation models, based on cellular automata and social network technologies, respectively identified as CAModel and SNModel. Both models were applied to the domain of innovation diffusion. More specifically, the theory of individual-based 'threshold' in
innovation diffusion, detailed in Edwards et al. (2003), has been implemented using CAModel and SNModel. The threshold theory considers that an individual adopts a given behavior according to a trade-off between social influences, measured by the number of neighbors adopting the same behavior, and a personal interest or resistance to change, known as the threshold.

CAModel and SNModel are characterised by the following:

- Both models are defined by a given constant number N of agents.
- Agents have the choice between behaviors A and B. Each time step corresponds to a re-evaluation of each individual's behavior.
- Each agent knows a number of other agents (its neighbors).
- The CAModel considers that each agent has four neighbors, associated to their position (north, south, east and west). All cells are occupied. If an agent knows another one (a neighbor), this one also knows the first agent as its neighbor.
- The SNModel considers that each agent has its own social network. The social network of an agent is composed exactly of four neighbors, for preserving comparability with the CAModel. Neighbors are determined randomly in the setup of the simulation. Social network links are unidirectional, meaning that if an agent knows another agent, this one might not know the agent. This depicts a fundamental distinction between both models: SNModel allows representing the notion of non-reciprocal social influence; which is not possible with a cellular automata-like model, such the CAModel.

In both models, in each time step each agent computes the utility of adopting A or B. $a, b, c, d$ are parameters reinforcing the positive or negative influence of neighbors. $V(i, A)$ and $V(i, B)$ represent the number of neighbours of agent $i$, concerning respectively behaviour A or B. The utility $U_i(A)$ and $U_i(B)$ of agent $i$ to choose A or B is expressed as (Edwards et al. 2003):

$$U_i(A) = a \cdot V(i, A) + c \cdot V(i, B) + e_i^4 \quad \text{and} \quad U_i(B) = d \cdot V(i, A) + b \cdot V(i, B) + e_i^\beta$$

Stochastic events will also influence the behavior of an individual. The uncertainty in the decision is ruled by the parameter $\beta$. The probability of adopting behavior A or B is computed by the following:

$$P(i \text{ chooses } A) = \frac{\exp(\beta \cdot U_i(A))}{\exp(\beta \cdot U_i(A)) + \exp(\beta \cdot U_i(B))}$$

$$P(i \text{ chooses } B) = \frac{\exp(\beta \cdot U_i(B))}{\exp(\beta \cdot U_i(A)) + \exp(\beta \cdot U_i(B))}$$

(see Edwards et al. (2003) for a detailed explanation of formulas).
The theory of an individual-based 'threshold' in innovation diffusion was a pretext to model, implement and test two distinct models of social simulation. The implementation of a CAModel, based on the cellular automata paradigm, has been tested using the parameter test sets referred to in Edwards et al. (2003). The results were characterized by close proximity to cyclic periodicity. The following graphs depict the total number of behaviours A and B through time, for two test sets of parameters:

Figure 5. Total number of choices A and B, to the parameter test set $a=1$, $b=2$, $\beta=0.15$, $b=8$ and $c=6$, concerning implementation of CAModel (left) and SNModel (right)

Figure 6. Total number of choices A and B, to the parameter test set $a=1$, $b=7$, $\beta=0.15$, $b=9$ and $c=4$, concerning implementation of CAModel (left) and SNModel (right)

Comparison of the results obtained through both models allows verification of the existence of longer periods in the CAModel. These results were obtained concurrently, permitting simultaneous comparisons relating to the same parameter test set. Figure 7 depicts the processor activity, showing the main simulation process and the two concurrent simulations, each one executed within its corresponding light process. The main process (reference "_Total") is responsible for the execution of the java virtual machine and the main execution process. References "javaw" and "javaw#1" concern the execution of the two concurrent simulation models.

Figure 7. Balance between the processors as the model is in execution
This example demonstrates some of the advantages of using concurrency, namely observing the evolution of the results, and using exactly the same parameter set that inputs both models at a common starting moment. Nevertheless, an important limitation was evidenced with this first experiment: we could have arrived at the same conclusions by just executing the models in a sequential order and comparing the results afterwards. This has motivated further experiments with potential advantages over the first experiment.

4.2. Second example – synchronization, coordination and composition of hybrid models

A second example has been designed, aiming to test the synchronization and coordination of concurrent simulations. Recalling the idea introduced in section 3.2 (see figure 4), concurrent synchronization of two distinct models allows data exchange and ascertaining the composition of hybrid models. This example, in particular, intends to test the composition of hybrid models.

The concept of synchronization point is here represented through the graphical representation of Petri Nets (Riemann, 1999). This mathematical formalism is particularly adapted to the representation of concurrent systems. A Petri Net is composed of place nodes (p), transition nodes (t), links between place nodes and transition nodes, and tokens. Links are unidirectional. Each transition can contain one or more tokens. A transition is fired if there exists at least one token in every place connected to the link. When a transition is fired, tokens are produced in every place leaving the transition. In each moment, the global state of the system is represented by the position of the tokens in each place.

Figure 8: The synchronization point between two complementary models
The figure 10 represents the concurrent execution of two abstract models, A and B. Let us consider that the base simulation environment defines the synchronization point represented by the transition $t_0$. In the left, model A has advanced its execution and is available to fire the transition $t_0$. A waits for the availability of B. In the centre, models A and B are both ready to fire the transition $t_0$. At this moment, the transition point is active, meaning that both models are synchronized at the same point of their execution. At the synchronization point, the models can communicate; the base simulation environments can also evaluate the performance of each model up to the point. The intermediate evaluation of performance will allow for hybrid models, composed by different models at each stage.

The general concept of the synchronization point between concurrent simulations has been tested in the domain of innovation diffusion. To maintain some coherence with the previous example, we have considered the theory of individual-based 'threshold' (Edwards et al. 2003). There have been three adaptations of this theory. The first model is based on the notion of epidemic models, introduced by Antonelli (1992) and developed further by Solomon (2000), using percolation models from physics, to model the dynamics of behavior adoption. In this model, the propagation of innovation processes is random. At each interaction of the simulation, each agent propagates his opinion to a random number of agents between three and five. The second model is an evolution of the first one, with more structured diffusion processes (Pyka and Fagiolo, 2005). Each agent has a contact network that he uses to propagate his opinions. Each network has four agents. Contact networks are redesigned at each iteration, where one agent is replaced by another agent chosen randomly. Finally, the third model considers that each agent has a contact network and propagates his opinions only to his contacts. In this last model, networks are static. For example, they are not changed during the simulation (Cowan, 2004). This way, the three models stand from a dynamic model, with random propagation, to a static model with network propagation. Let us consider the execution of the three models.

<table>
<thead>
<tr>
<th>Random epidemic model</th>
<th>Dynamic network model</th>
<th>Static network model</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_0$</td>
<td>$P_0$</td>
<td>$p_0$</td>
</tr>
<tr>
<td>1st phase</td>
<td>t_0 is fired</td>
<td>2nd phase</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$P_3$</td>
<td>$P_3$</td>
</tr>
<tr>
<td>3rd phase</td>
<td>$t_1$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 9: Concurrent execution of the three models
Figure 9 shows transition $t_0$ ready to be fired, meaning that the base simulation environment can evaluate the quality of each model until that point. Next, the simulations continue and are evaluated at each synchronization point, that is, at the end of each phase. This mechanism allows for partial evaluation of the models, according to the specific characteristics of each phase.

Let us now consider that we want to simulate the knowledge diffusion process concerning a particular innovation in a given society. We have used some of the results of the INNOVANET project, conducted in the spring and summer of 2003 that included interviews with representatives from industry, namely the innovation knowledge life cycle proposed by Paukert et al. (2003). The beginning a diffusion process is characterized by non-structured propagation, because organizations are not yet ready to integrate the innovation within the production process. This way, the first stage of the life cycle corresponds to the diffusion by community knowledge. In the next stage, organizations start to appropriate the innovation. The diffusion process is partially structured, through both knowledge and professional networks. This stage is characterized by organizational knowledge. Finally, the last stage of the innovation diffusion process is characterized by the existence of static networks, that organizations use to control information and to integrate innovation into their activities and processes. This last stage is known as working knowledge.

The three models referred to above can be applied to the innovation knowledge life cycle through a hybrid simulation model, in the following way:

![Figure 10: Hybrid model composed by three different models](image-url)
The Random Epidemic Model is used in the stage of community knowledge, where the propagation of knowledge is unstructured and takes place through individuals. The Dynamic Network Model is adapted to the organizational knowledge stage, because the diffusion process is semi-structured, with coexisting knowledge and professional networks. The Static Network Model corresponds to the last stage of working knowledge, where organizations control information. This example illustrates the way in which a hybrid model can explore the complementarities of its components, obtaining a simulation model adapted to the reality being studied.

5. Discussion

An important advantage of this approach is the possibility for the modeller to consider run-time interaction within a set of models. This improves the usual sequential analysis and facilitates comparison of social simulation models. In reality, establishing synchronization points provides a dynamic way for resource sharing and task coordination, throughout the concurrent execution of different models. These characteristics might lead to the composition of hybrid models, particularly adapted to specific social environments and situations. Nevertheless, a run-time comparison is not always necessary or even desirable. Frequently, models can be compared sequentially. On the other hand, concurrent comparison is more complex to set up.

Another significant issue concerning the approach relates to the performance of social simulations. Concurrent simulations will naturally evolve to simulations where agents are distributed over a computer network. This will permit the implementation of more sophisticated and heavier agents, comprised in social simulations between cognitive agents.

6. Further developments

Comparing and evaluating simulation models has the important advantage of facilitating model transfer between researchers. In this sense, an extension to the propositions presented in this paper will be to contribute to the creation of a typology of models. This development will possibly take the form of a model library, available on the Internet and composed dynamically using the diversity of contributions from the research community. This development will be an addition to other recent contributions (Cioffi-Revilla et al. 2003).

On the other hand, some degree of standardization concerning the forms of representation is needed to allow model replication and validation, in the sense of Axelrod (1997). Some formal language, able to identify the models’ characteristics and generally accepted by the scientific community, is needed. These requirements will hopefully be (partially) satisfied by the results of recent ongoing research projects, such as Diversity (2006), dealing with the adoption of graphical formalisms, like UML or Petri Nets, to MABS.

These two directions aiming to improve the ideas proposed in this paper face several difficulties: (1) the recognition by the scientific community of a MABS typology; (2) the
adoption of a common framework of description for the models, simulation protocols and 
results (Rouchier 2003); and (3) the definition of a methodology to link some abstract 
formal specification of simulation models to their implementation and experimentation. 
These are some of the challenges that we will be facing over the next few years.

7. Conclusion

This paper proposes a generic architecture allowing concurrent execution of simulation 
models. We have implemented the architecture and carried out an experiments 
concerning the concurrent execution of alternative models, on the one hand based on 
cellular automata and social network approaches, and on the other hand based on 
previous research concerning the specific domain of innovation diffusion. This 
experiment showed some potential advantages of comparing models concurrently, such 
as performance evaluation at run-time and the possibility of interaction between 
complementary models, making possible the composition of hybrid models.

8. References

Concurrent Programming”. ACM Computing Survey 1983, volume 15, number 1, 
pp.3-43. ACM Press.

Elsevier.

AXELROD, R. (1997) “Advancing the Art of Simulation in the Social Sciences”. In 
Conte R., Hegsekmann R. and Terna P. (Eds.) Simulating Social Phenomena, 

Development Kits: An Evaluation”. CNSR 2003 Communication Networks & 
Services Research, May 15-16.

social simulations: GeoSim and FEARLUS models”. Journal of Artificial 
Societies and Social Simulation vol. 6, no. 4.

DIVERSITY (2006), research project description available on-line at 
http://iscte.pt/~jmal/DIVERSITY/

Multi-agent Platform.”. Agent-Oriented Software Engineering III. Third 
International Workshop, Agent-oriented Software Engineering (AOSE), Bologna, 

individual-based model of behaviour diffusion with its mean field aggregate 
approximation”. Journal of Artificial Societies and Social Simulation vol. 6, no. 4.


