Towards an Emergence-Driven Software Process for Agent-Based Simulation

Nuno David[1,2,‡], Jaime Simão Sichman[2,¥] and Helder Coelho[3]

[1] Department of Information Science and Technology, ISCTE/DCTI, Lisbon, Portugal
Nuno.David@iscte.pt http://www.iscte.pt/~nmcd
[2] Intelligent Techniques Laboratory, University of São Paulo, Brazil
Jaime.Sichman@poli.usp.br http://www.pcs.usp.br/~jaime
[3] Department of Informatics, University of Lisbon, Portugal
hcoelho@di.fc.ul.pt http://www.di.fc.ul.pt/~hcoelho

Abstract. In this paper we propose an emergence-driven software process for agent-based simulation that clarifies the traceability of micro and macro observations to micro and macro specifications in agent-based models. We use the concept of hyperstructures [1] to illustrate how micro and macro specifications interact in agent-based models, and we show that the reductionism/non-reductionism debate is important to understand the reliability of agent-based simulations. In particular, we show that the effort expended in the verification of agent-based simulations increases exponentially with the number of micro and macro specifications, and that the reliability assessment of non-anticipated results in simulation is in practice not possible. According to these results we claim to be impossible in practice to verify that an agent-based conceptual model has been implemented properly as a computational model, since one does not usually know what is the desired output a priori. We thus advocate that the classic process of verification, validation and exploration of non-anticipated results is not reliable in an agent-based simulation, and call into question the applicability of traditional software engineering methods to agent-based simulation.

1 Introduction

The software process is the set of activities and results that produce a software product. In Software Engineering the attributes of a software product refer to the *non-functional* characteristics displayed by the product once it is installed and put to use. These attributes characterize the product's dynamic behaviour and the use made of the product, while reliability and usability are among the most fundamental traits (see [19]).

Meanwhile, the Agent-Based Simulation product differs in various senses from the classical one. Similarly, the process of product development in **A**gent-**B**ased

‡ Partially supported by FCT/PRAXIS XXI, Portugal, grant number BD/21595/99.
¥ Partially supported by CNPq, Brazil, grant number 301041/95-4, and by project MAPPEL (PROTEM -CC, CNPq/NSF), grant number 680033/99-8.

Simulation (ABS) proceeds from different motivations from the ones originating in classical process. By *ABS product* we mean the set of programs and documents required to satisfy the designer and users' goals of running successful and informative simulations. We identify four fundamental aspects that characterize the ABS product:

i) the product is instantiated as such after the first development cycles, and throughout the implementation phase with the first simulation experiment;

ii) the product does not stipulate an exhaustive and pre-specified enumeration of requirements that it must satisfy once it is put to use; instead, a model is developed along succeeding cycles that simultaneously include specification, implementation, verification, validation and use; the motto is thus *exploration of requirements*;

iii) the product is frequently used to *explore outputs* that are difficult to calculate analytically from complex models;

iv) the product is normally used to *explore qualitative concepts* according to outputs that are neither anticipated nor intentionally specified during the model specification phase, which makes the classical notion of *dynamic verification*[1] very difficult to apply.

Characteristics (i) and (ii) have been well identified and systematized problems in classical **S**oftware **E**ngineering (SE) and **A**rtificial **I**ntelligence (AI), particularly in the field of exploratory programming for knowledge-based systems. Typical processes to develop these products are strongly based on formal or informal verification of programs according to requirements that must be specified at some point in the development process. Contemporary models of **A**gent **O**riented **S**oftware **E**ngineering (AOSE) (see [4]) do not add novelty in this respect, for its methodologies concern the interaction of distributed intelligence, so as to define behaviours and solve problems that must be specified at some point of the development process.

Characteristic (iii) has been reasonably systematized in classic **C**omputer **S**imulation (CS) for dynamic system analysis, queuing models or general-purpose engineering (see [13]). In this case, the exploration of requirements and results is more quantitative than qualitative in nature. Most model specifications come in the form of equations, and the verification process usually arises in the form of mathematical analysis.

Characteristic (iv) is potentially more defining of ABS products. It results not only from the product role of *exploring requirements* and *program outputs*, but also from *exploring qualitative concepts* according to such outputs. In effect, a relevant use of ABS is to detect which effects may be drawn from patterns of agent interaction, that is, the emergence of macro-level regularities that are not intentionally specified at micro levels of detail. These observations must be verified *a posteriori*, after program execution trials, and some of them may be hard to verify and validate. Hence, in most cases, the development of ABS products covers additional levels of difficulty if

[1] The role of verification is to show that the computational model is equivalent to the conceptual model. *Dynamic* verification checks program correctness by evaluating given inputs with program execution outputs (see §2, [17,18,19] and [21] in this volume).

compared with classical products, since the borders between *dynamic verification* and *dynamic validation*[2] are harder to distinguish.

Methodological issues in ABS have been a topic of primary importance in the discipline (see e.g.[5,9,20]). Meanwhile, the research has rarely approached the technical details of software processes and its relationship with the problematic of emergence. While the software process may have been somewhat considered by some authors, the underlying methodological analysis has been invariably understood according to assumptions and principles of classic SE, classic CS, AI and AOSE, whose feasibility of purpose in ABS is far from being clear. At this stage, it is important to include the software process in the ABS research agenda for the following reasons:

i) the ABS product is not only used for technological automation of problem solving and expertise support, but also for purposes of scientific investigation of social theories and models of the real world. While in the former case the results of ABS may be reasonably verified with specifications and are most times validated by its users' *desires*, in the latter its validation requires independence from the users' and developers' bias;

ii) the crucial role of non-anticipated observation of program execution outputs in ABS hinders the applicability of classic software development principles to ABS. One must remember that validating a conceptual model does not guarantee the correctness and acceptability of non-anticipated outputs in the corresponding computational model. Since in ABS the verification and validation of non-anticipated outputs are superimposed, the existence of development premises for ABS that have not been considered in classic SE, AI and AOSE can be hypothesised. This change of premises may have a profound impact on the product non-functional requirements, especially, with respect to reliability requirements;

iii) most classic CS and AOSE products are not concerned with emergent, non-anticipated, structures. Classic CS is not agent-based. Organisations and institutions specified in AOSE are both considered as computationally individual entities in the system; here, the interaction between different levels of agent granularity is usually pre-defined; the concept of emergent "wholes", multi-level modelling, and emergent interaction between agents at different levels of abstraction is usually undesirable. The specification of models in ABS must thus be significantly different from classical CS and AOSE. Currently, there are no meta-models to describe the ABS process. For this reason, the methodological and epistemological tension between individualistic and holistic approaches has been discussed from an abstract point of view, with no real connection to practical ways of specifying agent-based simulations. The inexistence of meta-models to specify software is an excellent way to attain incorrect implementations and bad verification and validation. Indeed, specifications need to be *traceable* to implementations and program execution outputs.

---

[2] The role of validation is to show that the conceptual model specification is equivalent to an intended application or observed target. In practice the computational model must be validated as well. *Dynamic* validation checks if the computational model outputs agree with the intended application or observed target (see §2, [17,18,19] and [21] in this volume).

In this paper we investigate an abstract model that characterizes the ABS software process with regard to the specification stage. We concentrate on the logical analysis of the method *per se*, rather than validate the discipline through simulation results and case studies. Method examination facilitates detection of inconsistencies and errors related to problem conceptualisation and program construction. It is important to establish systematic development and interpretation principles, particularly with respect to the analysis of emergent phenomena.

This paper is structured as follows. In section 2 we will analyse the role of conceptual and computational models in agent-based simulation. Later we will show how to incorporate micro- and macro-observations in agent-based conceptual models, and also show that the reductionism/non-reductionism debate is important to understand the reliability of simulation programs. In particular, we will demonstrate that the effort expended in static verification increases with the number of micro and macro observations specified in agent-based models, and that an extensive verification of anticipated results in agent-based simulation is in practice not possible. In section 3 we will use this result to assert that the process of dynamic verification, validation and exploration of non-anticipated results is not reliable in agent-based simulation. We will point out research directions to solve this problem and give our conclusions in section 4.

2 From Conceptual to Computational Models

It is relatively unanimous that the ABS process begins with the identification of some research object, a "puzzle", for which there are questions whose answer is not known (cf. [11]). The research object leads us to the definition of an abstract or real world *target* for modelling. The target is the real or proposed idea, situation or phenomena to be modelled, from which one constructs a *conceptual model* (a mathematical/logical/verbal representation) based on observations, assumptions and formulation of hypothesis. The conceptual model must be transformed into a corresponding *computational model*.

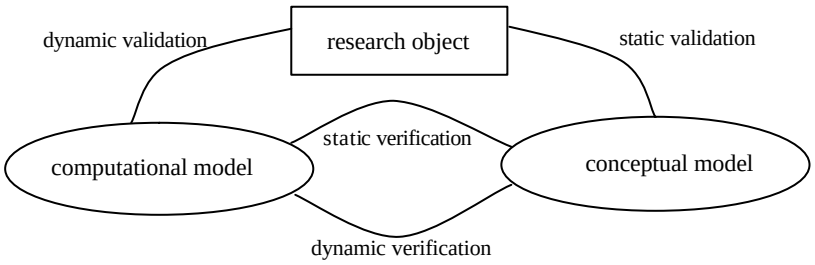

Figure 1 – The modelling process with relation to verification and validation.

Consider the simplified version of the agent-based modelling process in figure 1. The figure relates verification and validation with the modelling process (see [18,21]). *Static validation* is defined as determining that the theories and assumptions underlying the conceptual model are reasonable representations of the research object.

Static verification is defined as ensuring that the program source code representing the conceptual model is correct. Therefore, static verification is not directly concerned with the evaluation of non-anticipated outputs during the simulation. *Dynamic verification* is defined as determining that the conceptual model entails the computational model outputs. *Dynamic validation* is defined as determining that the computational model outputs are correct according to the research object[3]. In the present section we investigate the stages involving the specification of conceptual models and static verification.

2.1 Multiple Targets in Agent-Based Simulation

The particularity of agent-based modelling is the definition of elements in a model that represent entities in the target, such as human beings, organizations or inanimate objects. Eventually, every passive and active entity in the target may be "agentified". Some approaches differ in respect to the model's level of abstraction in relation to the real world, as well as to the representation granularity of agents in the model:

i) *simulation with artificial societies* - at one extreme of the abstraction axis we find models of artificial societies that do not reference a concrete target or specific theory about the real world (e.g.[10]). This approach is most akin to requirements exploration in classical SE and AI: the conceptual model is validated according to the designer's *observation perspectives* and *desires* in respect to the computational model outputs. The conceptual and computational models evolve in consecutive refinement cycles according to those judgments. Hence, a major part of the software process is devoted to the *verification* of models: to what extent are the macroscopic regularities of interest given by the computational model outputs caused by the local or micro mechanisms specified in the conceptual model? This tendency often suggests an implicit assumption that does not have to be adopted, when the term *micro* is associated with specifications of conceptual and computational models, and the term *macro* with observations of computational models outputs.

ii) *animation of socio-cognitive theories* - halfway to the other extreme of the axis, we find the simulation of models of socio-cognitive or sociological theories. This approach is usually founded on computational animation of logic formalisms, which represent agents and social structures according to a specific theory, for instance, the Theory of Dependence and Social Power [3]. The animation serves a purpose of theory consistency checking and refinement of social theories (see [6]), as well as to verify the pragmatic feasibility of such theories in MAS engineering [22,8]. In this trend, the structures emerging from the interaction of elements in the model are crucial for theory refinement. By emergent structures we mean new observable categories that do not find a semantic expression in the original vocabulary of the model (e.g. the subjective observation of agent *coalitions* according to patterns of *dependence relations*). Recent methodological discussions propose the explicit specification of given emergent (macro) entities in models, regardless of being

[3] In the software engineering literature, dynamic verification and validation is also known as "program testing".

reducible or not reducible to levels of description at lower granularity (see, e.g., Sawyer's claim in [5] or [27]). As we will soon show, such practice has always been common in ABS, but it has not been methodologically and appropriately assumed.

iii) *simulations with representations of the real world* - at the end of the axis, we find simulations with models that should desirably represent observations of real social and institutional concrete processes (see, e.g., Moss' claim in [5]). The goal is "the use of MAS to describe observed social systems or aspects thereof and to capture the sometimes conflicting perceptions of the social system by stakeholders and other domain experts". This approach is the most conflicting with the classic process, owing to the practical inevitability of defects in computer programs. Insofar as specification and program coding errors are unavoidable in practice, the *validation* of computational model outputs is susceptible to misjudgements because the *verification* and *validation* processes overlap with respect to the evaluation of such outputs.

We can see that when travelling from approach (i) to (iii) there is an increasing overlap of dynamic verification and validation judgements. Such increasing overlap also parallelizes a change in the scientific role of simulation from *explanation and prediction* to *explanation and representation* of targets. But as we will see, the complexity of the verification process in the former case, and the superposition of verification and validation judgements in the latter, obstructs the feasibility of SE techniques in ABS.

2.2 Micro or Macro?

One may say that in most ABS approaches there is an association flavour of the term *micro* with the specification of conceptual and computational models, and the term *macro* with the computational model outcomes. But this is not entirely factual. There are at least two cases for which the term macro is underlying the specification process:

i) *implicit in the target* – there are elements in the universe of the model that are indivisible, but represent entities of the target with a higher (macro) [or lower (micro)] granularity than the other indivisible elements with whom the element interacts in the model.

ii) *explicit in the model* – there are elements in the universe of the model that represent aggregates of other indivisible or non-indivisible elements, that is, a set of elements defined in the model as a "whole", having specific, and possibly exclusive, properties.

In the first case the macro concept is subjectively relevant with respect to the target, but irrelevant to the model objective dynamics. In the second case, there are aggregates treated explicitly as "wholes" in the model. Some researchers would agree that the most frequent approach is the first one. However, there are numerous examples of the second, e.g., in Swarm [12] agents may be contained in *swarms* that are in turn contained in other *swarms*. The source of this question and its effects on

the product reliability requirements can be found if we further abstractly detail the specification of agent-based models.

2.3 Agentified Conceptual Specifications

We start the specification of a model with its set of indivisible agents. This set will be known as the *Agentified Conceptual Specification* (ACS). An ACS is a set of *Agentified Entities* (AEs) associated with a finite index set $\boldsymbol{I}$, $ACS=\{A_i\}, i \hat{I} \boldsymbol{I}$.

An AE is an element in the model specified according to some arbitrary observation process. The role of the observation is to analyse a particular feature of abstract or physical nature in the target. Different AEs can represent entities of different granularity in the target, such as physical objects, humans, organisations, realistic representations of organisational concepts (e.g. a norm). Before we proceed, it should be clear for the sake of generality that it is indeed possible to represent any feature of a target by agentifying it, despite the disadvantage of overstressing the agent metaphor (see [20]).

At this point we have not represented explicit macro concepts in the model. The specification of agentified aggregates can be accomplished with observation processes that explicitly describe macro-observations of the target.

2.3.1 Levels of Observation and Emergence: An Hyperstructural Vision

The ABS product development is an exploratory and iterative process, for which the conceptual model is to a great extent determined by observing properties of entities in the target, and how the entities relate and interact with each other according to those properties. Particularly relevant to ABS is the observation of entities according to different levels of abstraction and perspective. We view an ABS model as a hyperstructural construction [1,26]. The hyperstructural framework is useful to model AEs that can interact with other AEs at different levels of abstraction. Given one or more arbitrary observation processes we define two qualitative distinct levels of observation, called *micro-observation* and *macro-observation*:

Micro-observation – the observation of properties of each individual AE in the target:

$$Obs(A_i), i \hat{I} \boldsymbol{I}$$

Macro-observation – the observation of properties of "wholes", according to (i) any subset $\boldsymbol{K} \hat{I} P(\boldsymbol{I})$ of AEs in the ACS; (ii) micro-observation of those AEs; (iii) an interaction process between those AEs using properties observed in (ii):

$$Obs(A_{\boldsymbol{K}}), with\ A_{\boldsymbol{K}}=R(A_j, Obs(A_j), Int)), j \hat{I} \boldsymbol{K}, \boldsymbol{K} \hat{I} P(\boldsymbol{I})$$

where every whole $A_{\boldsymbol{K}}$ is indexed by some element $\boldsymbol{K} \hat{I} P(\boldsymbol{I})$.

The conceptualisation of a new structure, a "whole", involves observing an aggregate of AEs in the target, where *Int* is an interaction process between those AEs and *R* is the result of the construction process. For instance, consider an aggregate of AEs characterized by a macro property, say the gross income. The micro-properties are the AEs' individual incomes, and the construction process is an aggregation of AEs according to an interaction process that calculates the gross income based on the individual incomes. In other cases, the observed macro-properties may be disjoint

from the observed micro-properties, for instance, the *cohesion* of an aggregate calculated according to patterns of relations of trust and dependence between its members.

If we want to represent explicit macro-observations in the model, we have to specify a new type of AEs in the ACS. The specification of such wholes is achieved by representing new AEs at higher levels of granularity than their constituent AEs. A structure A_K denotes a *first order macro-AE*, conceptualised in a distinct level of abstraction from the other *micro-AEs*. Thus, in the previous example, the concept *cohesion of an aggregate* would be specified as a new macro-AE in the conceptual model. Nevertheless, note that the construction process defining the macro-AE does not have to be expressed in the specification. Moreover, it can be argued that the observation of macro-properties does not even need to be computationally expressible, i.e., by way of an algorithm. Indeed, like any other micro-AE, the designer can ascribe (by definition) additional macro-properties to a macro-AE. For instance, if one specifies in the model an organisation that contains several agents, one is not necessarily assuming that those agents are sufficient to specify the properties of that organisation. If that is the case, the micro-observations and the interaction process (the construction process R) are thus subjectively represented in the observer's mind. *It is common practice in ABS to define properties of macro-AEs without stating its hypothetic reducibility to properties of its constituent AEs according to a particular theory. Such omission is perfectly acceptable, but it should inhibit the designer from ascribing methodological individualistic principles to his simulation.*

Sawyer [5] is right when he claims that the inspiration principles of ABS are methodological individualistic, but his claim that most agent-based models are methodological individualistic seems to be false; in practice most models represent macro-concepts, regardless of being reducible or non-reducible to other micro-concepts. Nevertheless, as we will soon show, Moss [5] does not seem to be right in suggesting that the reductionism/non-reductionism debate does not influence the verification and validation problem in ABS.

The set of observed micro-AES and first-order macro-AEs constitute a *first-order ACS*, denoted by ACS^1. The maximum number of elements in a first-order ACS is given by $\#ACS^1 = \#P(ACS) - 1$. It is evidently possible to consider second order macro observations over first order ACSs, and so on. The process that combines AEs of different order is called a hyperstructure, and is illustrated in figure 2.

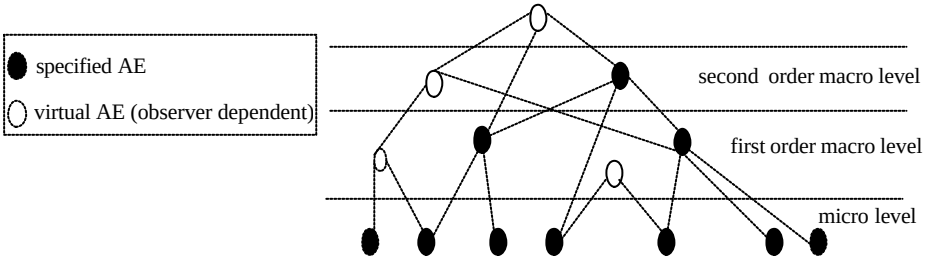

Figure 2 – Specified AEs and virtual emergent AEs. The convergence of lines from lower order to higher order AEs means that lower order AEs interact with one another. This may be ascribed by the specification itself or be only represented in the observer's mind. For instance,

the notion of observed 'system" in a simulation, e.g., the construction process given by the interaction between a grid and a set of individual agents, can be understood as a virtual AE; in such example the concept of 'grid' is thus agentified.

We proceed from the principle that the interesting emergent events that involve agent-based simulations reside not only in the simulations themselves but in the ways that they change the way we think and interact with the world. "Rather than emergent devices in their own right, computer simulations are catalysts for emergent processes in our mind" [2]; they help us to create new ways of seeing the research object. In the figure, the black vertexes denote AEs that are explicitly specified in a model. The white vertexes denote possible observable *virtual* AEs, which are not part of the specification. Some of them will constitute the observer's expectations of how the model will behave; consequently, *they must be verified during the software process*. We immediately see that the observer's interest can be the conceptualisation of ACSs of higher order than the one specified, for example, the conceptualisation of the system as whole.

2.3.2 Specifying State Transitions

The ACS modifies its state along the time by modifying the state of its constituent AEs. We denote an arbitrary order ACS in state e by $\boldsymbol{A}_{|e}$, where $\boldsymbol{A}_{|e}=\{A_{i|e}\}$, $i \hat{I} \boldsymbol{I}_{|e}$ and $\boldsymbol{I}_{|e}$ is the index set in state e. The notation $A_{i|e}$ denotes an AE with index $i \hat{I} \boldsymbol{I}_{|e}$ in state e.

Consider a set of properties given by some arbitrary observation process *Obs* over the AEs in the ACS $\boldsymbol{A}_{|e}$. A state transition $e1 \rightarrow e2$ in the simulation is calculated by submitting the AEs in state $e1$ to an algorithmic process of interaction *AlgInt* that uses the properties registered under the observation process *Obs*. A transition sequence $e1 \rightarrow e2 \rightarrow ... \rightarrow en$ is obtained by successive compositions of the algorithm $\boldsymbol{A}_{|en}=AlgInt_{Obs|e1 \rightarrow en}(\boldsymbol{A}_{|e1})$.

The specification of the interaction algorithm will have a decisive impact in the AEs' evolution. For instance, new AEs may be created while others will cease to exist. It is also possible to specify mechanics to observe aggregates composed of AEs at different time states. These observations would allow the agents themselves to establish the *persistence* in time of some regularity. Such observations would have to be *previously* and *algorithmically* specified, which is different from the usual designer-observer's position that *mentally* conceptualizes the emergence of *new* categories *during* or *after* the simulation.

2.4 Static Verification

The need to transform conceptual models into equivalent computational models has always been a major motivation for the substratum of software engineering techniques. Verification techniques involve showing that the detailed design and implementation of the conceptual and computational models are correct. It involves static and dynamic techniques (see [17,18,19]). Static techniques are concerned with the analysis and checking of system representations such as specification documents and the program source code. *Static techniques are not directly concerned with*

evaluation of non-anticipated emergent outcomes during the simulation. For this reason, some static techniques used in classical SE and AOSE can also be used in ABS, although it will probably involve in most cases a higher degree of complexity analysis.

Within an agent-based realm of complexity, static verification involves to a great extent checking the model specification with its detailed design. There is an increasing amount of literature for verifying agent-based systems in AOSE. One recent approach is static compositional verification [7]. For macro-properties assumedly reducible to lower order properties, this involves verifying that the properties of macro-AEs derive from its constituent micro-AEs properties, according to some interaction algorithm.

Consider a *m*-order ACS $\boldsymbol{A}_{|e1}$ with an index set $\boldsymbol{I}_{|e1}$. The state *e1* is the initial state. Within a hyperstructural framework, verifying the model *expected* behaviour can be described by the following steps:

(i) verify *specified* AEs by composition: given the properties of micro-AEs, verify that the properties of each macro-AE resemble the specification of the macro-observation, i.e., for each *j*-order macro-AE $A_{i|e1}\,\hat{I}\,\boldsymbol{A}_{|e1}$ and $j£m$, show that there is a subset $\boldsymbol{I_i}\,\hat{I}\,\boldsymbol{I}_{|e1}$ of AEs with order less or equal to *j-1*, such that $A_{i|e1}=R(A_{|e1},Obs(A_{|e1}),Int))$, $l\,\hat{I}\,\boldsymbol{I_i}$. So if A_i is an AE representing the concept *cohesion of an aggregate*, this amounts to show that the algorithm that computes the cohesion according to the set of lower order AEs conforms to its specification *R*;

(ii) verify the interaction algorithm *AlgInt*: given the properties of micro and macro AEs in state *e1*, verify that for any state *en* subsequent to *e1*, the properties in *e1* are preserved in *en*;

(iii) verify *expected* emergent AEs by composition: given the properties verified in (i), verify that the system will behave according to the designer's expectations. These are emergent properties associated with *virtual* AEs, possibly, with order greater than *m*. So if we specify that the system should reach some form of equilibrium after some state *en*, this amounts to show that it will do so according to the properties verified in (i).

The specification of macro-AEs is obviously a complexifying factor in the verification process. But the increase in complexity does not only lie in the compositional verification of step (i). Step (iii) implies the compositional verification of virtual aggregates, i.e., verifying expected emergent properties in the system. The number of observable virtual AEs will possibly increase with the number of micro- and macro-AEs specified in the model. For an ACS or order *m*, the highest number of observable virtual macro-AEs at order *m+1* is:

$$N_{virtual}\,(m) = 2^{n+n_1+\ldots+n_m} - n - n_1 - \ldots - n_m - 1$$

where *n* is the number of micro-AEs and n_k denotes the number of *k*-order AEs.

Thus, for a program without macro-AEs that manipulates 15 micro-AEs, the highest number of observable virtual AEs is 32753. If we include in the ACS one additional micro-AE and two second-order AEs the number increases to 262125. Of

course, the number of virtual AEs that must be statically verified is much lower. This is the case if the goal of ABS is the *exploration* of emergent events rather than the *specification* of emergent events. What this formula tells us is that even if we limit the number of observation processes to observe a simulation (limit the set of observable properties), the increase in the number of micro and macro-AEs in the model implies an exponential increase of observable virtual macro-AEs in the simulation. But if they are indeed observable, can we verify them? And if not, can they be validated without being verified?

3 Exploration of Results

We have now reached the conditions to confirm our claims in respect to assumptions in the classical process that cannot be used in the ABS process. For this purpose we make a short circuit in the development process, jumping directly to stages of dynamic verification, dynamic validation and exploration of results. It should be clear that we left behind many development steps that we have not approached here, such as static validation, detailed design and implementation.

For the time being, the unstated tendency in ABS to associate the term micro with the conceptual and computational model specifications, and the term macro with the computational model outcomes should now have been understood. For this reason, it should not be strange to see that the traditional attempt in AOSE to arrive at feasible verification processes is also implicitly adopted in ABS. The rule of thumb in AOSE, "specify few agents so as to get controlled and verifiable systems" is a particular case of "specify few micro and macro concepts so as to get controlled and verifiable systems". Surely, the ABS research community will hardly adopt such rule of thumb, and we would like to stay that way.

3.1 Dynamic Verification and Validation

It is often alleged that one fundamental difference that distinguishes the classical from the ABS process is the need in ABS to validate models with data and/or observations from real world targets. On one hand, the correctness of simulation results that are previously anticipated along the model specification phase is assessed with verification techniques that are similar, to a great extent, to classical SE and AOSE. On the other hand, the emergent tendencies that are not anticipated or intentionally specified are validated with data and observations of the target. The logic underlying this strategy is that if a program is correctly verified then its outputs are entailed by the conceptual model specification. This assertion would in fact be correct if we could rigorously verify the correctness of reasonable complex code.

What is the role of static and dynamic verification? Static verification techniques can partially demonstrate that the conceptual model corresponds to the computational model, but they cannot demonstrate that the computational model is indeed operationally useful and without software faults and failures. A *software failure* occurs when the software is executing, and the software does not behave as expected by the user. A *software failure* is not the same as a *software fault* (see [19]). *Software*

faults are programming errors whereby the program does not conform to the specification. Software faults are static and their existence may be inferred through program inspections (i.e. static verification) and from software failures. Software faults cause software failures when the faulty code is executed with a set of inputs that expose the software fault. For these reasons, dynamic testing is the predominant validation and verification technique in SE. Dynamic testing exercises the program using data like the real data processed by the program, and the values obtained are used to test if the computer program behaves as expected. It is used in both classical SE (see [19]) and CS (see [18]). The bad news for ABS is that real data is not easy to obtain. But even if that was not the case, suppose that the ABS program can be verified and validated for a set of known input/output relations. How can the reliability of such a simulation be affected after that stage?

3.2 Classic Software Engineering Methods are not Reliable in ABS

One result after the forty years of research history in software engineering is that any product that does not have its reliability requirements assessed cannot be validated with any model, whichever it may be. In most cases, classical products are ultimately verified and validated by the users' *desires* and *satisfaction* with the product behaviour. This means that if some dysfunctional result is not predicted before product delivery, then it must be uncovered with *acceptance tests*.

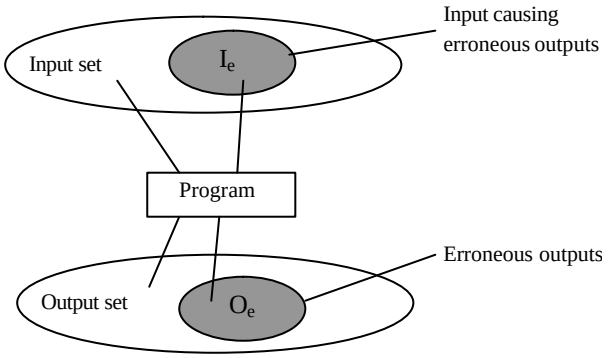

Figure 3 – Software faults cause software failures when the faulty code is executed with a set of inputs which expose the software fault. The software reliability is related to the probability that, in a particular execution of the program, the system input will be a member of the set of inputs which cause an erroneous output. In the classical product, there are usually a number of members of I_e which are more likely to be selected than others (adapted from [19], pp.351).

In fact, empirical data show that more than 50% of all effort expended on a program will be expended after it is delivered to the customer [17,pp.22;14;19]. Such procedures detect software failures, promoting the subsequent distinction between software faults (verification) and invalid specifications (validation). These findings result from empirical analyses that establish the following results (see [19]): (i) not all software faults are equally likely to cause software failures; (ii) transient failures or whose consequences are not serious to the users' needs are of little practical

importance in the operational use of the software. In some programs, empirical analysis find that removing 60% of product defects would only lead to a 3% reliability improvement [16]. Curiously, these are good results for classical SE. On the one hand, not all software faults imply less system reliability (if the probability of choosing an input which causes an erratic output is low – see figure 3). On the other hand, dysfunctional and annoying software failures can ultimately be detected with dynamic tests, including acceptance testing.

The difference between the classic and the ABS product can be founded precisely here. The first thing that we should be aware is that all simulation results are outcomes of a computational model, which is not necessarily equivalent to the conceptual model. Nevertheless, in the classical product the equivalence of the conceptual with the computational model does not need to be exhaustive. For the classical product, it is software failures not software faults that affect the reliability of a system [19,pp.357]. Hence, the existence of defects in the system is mostly assessed with dynamic and acceptance testing, according to the detection of unexpected outputs. During that stage, the emergence of very undesirable properties is suppressed through program correction or requirements re-specification. *The emergence of undesirable properties that are rare or not very annoying to users are typically not considered and not corrected.* As we have mentioned, removing product defects does not necessarily lead to reliability improvements. The emergence of properties that are irrelevant to users may not even be evaluated (actually, they will be rarely observed). To the users' concerns it does not matter if the system non-dysfunctional characteristics are, or are not, a result of a software fault. *Moreover, to the users' concerns it does not matter if some functional characteristic is, or is not, a result of a software fault.*

Now, dynamic and acceptance tests of this kind are definitely defective, and not appropriate, for ABS. This is due to the alteration of the product aims, where the users' satisfaction to accept the classical product is replaced in ABS with symbolic data and stakeholder (subjective) evaluation of social and institutional target realities. For the classic user, it does not matter if the conceptual model is not absolutely equivalent to the computational model, as long as the latter is functional to his desires and needs. Conversely, for the ABS designer and its users the goal is precisely the conceptual model, or/and a set of explanations according to the conceptual model. This may lead us to conclude that the ABS product requires high levels of reliability in much the same way as critical systems in classical SE, for the conceptual specification must be exhaustively equivalent to the computational model[4]. Of course, prescribing high quantitative reliability requirements to ABS programs is at present too much to ask. Furthermore, the nature of critical systems is what we do not need in ABS, since the goal of critical systems is precisely the complete suppression of non-anticipated results.

In reality, the problematic issue in ABS is more far reaching than reliability and correction *per se*. The problem is situated on the difficult definition of means to distinguish unexpected results from software faults and, therefore, to detect incorrectness. In effect, static verification in ABS is hard, as we have seen before;

[4] Critical systems are software systems that require high levels of reliability, like the software installed on a aircraft.

dynamic verification can contribute to enhance the product reliability according to a very limited range of input/outputs; there is no such thing as acceptance testing in ABS. *The distinction between non-anticipated outcomes, entailed by the conceptual model, and software faults in the computational model may, therefore, be virtually impossible to realize.* This indefinition is aggravated when sensitivity analysis and stochastic effects are at stake; for, to what extent is the conceptual model sensitive or not to software faults? Or to what extent is some stability and low sensitivity of results not a consequence of a software fault? These indefinitions call into question the applicability of classical SE techniques in the ABS development process. But do we have other techniques? Are we running too fast with the desirable increase in domain diversity, complexity and number of agents in simulations?

4 Discussion

4.1 Some Prospects on How to Address the Problem

In our vision, reliability is presently the fundamental problem in ABS. The problem of validation has been an important research issue, but are we adopting the right principles? Should we insist on the use of classic approaches and assumptions in regard to verification? We have demonstrated that we should not. Nevertheless, is there an alternative software process for ABS?

At the present maturing stage of ABS there is not yet an answer to these questions. But what this work has demonstrated is that the premises underlying program development and reliability of results have to be substantially revised. At this time such a need has not been incorporated into the discipline research trends. A careful reading of the literature of SE teaches us how the quantitative analysis of program construction has helped the discipline to find an equilibrium between theoretical and empirical knowledge, allowing a large scale production of programs with reasonable reliability patterns in a diversity of domains. Such effort may also be needed in ABS. For instance, are there typical patterns of code complexity in ABS programs? (see e.g. [15]). Is there a relation between the number of corrected software faults and an increase in reliability? As far as we know this type of quantitative analysis does not presently exist in ABS.

Other approaches might involve the so-called *alignment* of models [23], by comparing different models that announce the same type of results and try to see if they actually produce similar results (see e.g. [25] in this volume). There are some scalability limits to these approaches, however; particularly with respect to the level of code complexity (e.g. number of lines). The first is that it is highly biased to validate conceptual models, rather than to verify computational models. The second is that, as we have seen, a same conceptual model can lead to different computational models, which may produce similar or different outputs. The third is that it relies on a social process, the systematic process of re-implementation of programs, transfer of knowledge and assessment of results, which in the specific case of computational models, the implementations themselves, limits in practice the underlying inductive process of verification. In effect, it seems implausible to see programmers willing to collaborate on the massive inspection or re-implementation of programming code, not

only because it is a tedious and complex activity; as DeMillo *et al*. [24] would put it, static "(…) verification (of programs) cannot be internalized, transformed, generalized, used, connected to other disciplines, and eventually incorporated into a community consciousness. They cannot acquire gradual credibility, as a mathematical theorem does (…)".

The complementary approach that we have taken in this paper adopted a qualitative methodological course. Such effort allowed us to examine how the micro and macro modelling perspectives interact, affecting static verification and associated reliability requirements. This was possible because we have defined the concept of micro and macro specification in an unambiguous way. These results are also informative to the ongoing methodological and epistemological discussion, with respect to the tension between individualistic and holistic modelling trends in ABS.

4.2 Conclusions

In this paper we have analysed an abstract specification framework as part of an abstract development process for agent-based simulation. We have proposed an emergence-driven software process based on hyperstructures, which clarifies the traceability of micro and macro observations to micro and macro specifications in agent-based models. We have examined how micro and macro specifications interact in a model, and shown that the effort expended in the verification of observable emergent entities in simulation increases exponentially with the number of micro and macro specifications.

According to these results we have shown that it is impossible in practice to verify that an agent-based conceptual model has been implemented properly as a computational model, given that we do not usually know what we want the output to be a priori. Since in agent-based simulation the conceptual and the computational models must be exhaustively equivalent, these results demonstrate that the reliability assessment of non-anticipated results in simulation is in practice not possible. Furthermore, the results indicate that the reductionism/non-reductionism debate is important to understand the reliability of simulation programs. According to these results we have called into question the applicability of traditional software engineering methods to agent-based simulation. Hence, we claim that the premises underlying the applicability of classical program development to agent-based simulation have to be substantially revised.

References

1. Baas N.A. and Emmenche C. (1997). On Emergence and Explanation. In: *Intellectica*, 25, pp.67-83.
2. Cariani P. (1991). Emergence and Artificial Life. In: *Artificial Life II*, Addison Wesley, pp.775-797.
3. Castelfranchi C., Miceli M. and Cesta A. (1992). Dependence relations among autonomous agents, *Proceedings of MAAMAW92*, Elsevier Science, pp. 215-227.

4. Ciancarini P. and Wooldridge M. (eds), 2001. *Agent-Oriented Software Engineering*, Springer-Verlag, LNAI1957.
5. Conte R, Edmonds B., Moss S. and Sawyer R.K. (2001). Sociology and Social Theory in Agent Based Social Simulation:A Symposium. In *Computational and Mathematical Organization Theory*, 7(3), pp.183-205.
6. Conte R. and Sichman J.S. (1995). DEPNET: How to benefit from social dependence, In: *Journal of Mathematical Sociology*, 20(2-3), 161-177.
7. Cornelissen F., Jonker C.M. and Treur J. (2001). Compositional Verification of Knowledge-Based Systems: a Case Study for Diagnostic Reasoning. In *Series in Defeasible Reasoning and Uncertainty Management Systems*, vol.6, Kluwer Academic Publishers, pp. 65-82.
8. David N., Sichman J.S. and Coelho H. (2001). Agent-Based Social Simulation with Coalitions in Social Reasoning, In Moss S. and Davidsson P., editors, *Multi-Agent-Based Simulation*, Springer Verlag, LNAI, v.1979, pp.244-265.
9. Davidsson P. (2002). Agent Based Social Simulation: A Computer Science View, In *JASSS*, vol.5, no.1, http://jasss.soc.surrey.ac.uk/5/1/7.html.
10. Epstein J. and Axtell R. (1996). *Growing Artificial Societies: Social Science from the Bottom Up*, MIT press.
11. Gilbert N. and Troitzsch K. (1999). *Simulation for the Social Scientist*, Open University Press.
12. Langton C., Minar N., and Burkhart R., The Swarm Simulation System: A Tool for studying complex systems, http://www.swarm.org.
13. Law A. and Kelton W.D. (1991). *Simulating Modelling and Analysis*, McGraw-Hill.
14. Lienz B. and Swanson E. (1980). *Software Maintenance Management*, Addison-Wesley.
15. McCabe T. and Butler W. (1989). Design Complexity Measurement and Testing, In: *CACM*, vol.32, no.12.
16. Mills D., Dyer M. and Linger R. (1987). Cleanroom software engineering, In: *IEEE Software*, vol.4, no.2.
17. Pressman R. (1994). *Software Engineering: A Practitioner's Approach*, McGraw-Hill.
18. Sargent R.G. (1999). Validation and Verification of Simulation Models. In: *Winter Simulation Conference*, IEEE, Piscataway, NJ, 39-48.
19. Sommerville I. (1998). *Software Engineering*, Addison Wesley.
20. Troitzsch K.G., Brassel K, Mohring M. and Shumacher E. (1997). Can Agents Cover All the World?**,** In: *Simulating Social Phenomena*, Springer-Verlag, LNEMS 456, pp.55-72.
21. Marietto M.B., David N., Sichman J.S. and Coelho H. (2002). Requirements Analysis of Agent-Based Simulation Platforms: State of the Art and New Prospects. In this volume.
22. Sichman J.S. (1998). DEPINT: Dependence-based coalition formation in an open multi-agent scenario, In *Journal of Artificial Societies and Social Simulation*, 1 (2), http://www.soc.survey.ac.uk/JASSS/1/2/3.html.
23. Axtell R., Axelrod R., J.M. Epstein and M.D.Cohen (1996). Aligning Simulation Models: A Case Study and Results, *Computational and Mathematical Organization Th*eory 1(2), pp. 123-141.
24. DeMillo R., Lipton R. and Perlis A. (1979). Social Processes and proofs of theorems and programs, *Communications of the ACM* 22, 5 (May), 271-280.
25. Antunes L., Nobrega L. and Coelho H. (2002). BVG choice in Axerold's tribute model. In this volume.
26. Baas N.A. (1994). Emergence, Hierarchies and Hyperstructures. In Langton C. (eds), *Artificial Life III*, Santa Fe Studies in the Sciences of Complexity, Proceedings, Volume XVII, Addison-Wesley.
27. Sawyer R.K. (2001). Simulating Emergence and Downward Causation in Small Groups. In Moss S. and Davidsson P., editors, *Multi-Agent-Based Simulation*, Springer Verlag, LNAI, v.1979, pp.49-67.