

Agent-Based Simulation of Simulating Anticipatory Systems – Classification

Eugene Kindler

Abstract—The present paper is oriented to classification and application of agent technique in simulation of anticipatory systems, namely those that use simulation models for the aid of anticipation. The main ideas root in the fact that the best way for description of computer simulation models is the technique of describing the simulated system itself (and the translation into the computer code is provided as automatic), and that the anticipation itself is often nested.

Keywords—Agents, Anticipatory systems, Discrete event simulation, Simula, Taxonomy.

I. PROBLEMS IN PROGRAMMING OF COMPUTER SIMULATION MODELS AND THEIR SOLUTION

COMPUTER Simulation is the best (and may be said the unique) technique for exact studying of the complex dynamic systems, i.e. for the complex systems that are viewed to change their state during a Newtonian (non-relativistic) time flow. For studying dynamic systems that are not complex, simulation is a very expensive (therefore unfitting) technique.

Simulation is based on experimenting with the model [1]. In order to reflect the causality relations during the experimenting, the model is demanded to satisfy the following condition:

When events $E1$ and $E2$ coming in the model, respectively reflect events $e1$ and $e2$ coming in the modeled system, and when $e1$ occurs later than $e2$, then $E2$ cannot occur later than $E1$ in the model. In such a case, $E1$ usually comes later than $E2$, but it is possible that both the events come at the same time (it happens namely when the model is rough, but even such models are of use).

The reality of simulation is that the complex systems should be mapped to complex computer models with intricate behavior in time, which has to be described as algorithms handling with tangled structure of a lot of data. Moreover, a small change of the simulated system can be frequently discovered during the application of simulation; although such a change could be described by a simple sentence, the corresponding modification of the computer model is often so large that it is comparable with programming a new model from scratch.

In order to solve such problems concerning the programming of simulation models, an attempt was accepted and

Manuscript received June 27, 2006. The paper was elaborated under support of the grant No. 201/06/0612 of the Grant agency of Czech Republic.

Eugene Kindler is with Ostrava University Faculty of Sciences, Czech Republic (phone: +420-221-914-286; fax: +420-221-914-323; e-mail: evzen.kindler@mff.cuni.cz).

elaborated in two ways, which will be shortly described in the next lines; the attempt can be characterized so that instead forcing the author of simulation model M to describe what should happen in the computer C when M exists and operates inside C , the author should describe the simulated system and such a description should be automatically translated into the computer code realizing M . In the present paper, the attempt will be identified as Φ . The last fifty years of computer simulation showed an enormous help of it.

The first elaboration of the attempt consists in *simulation programming languages* (shortly *simulation languages*). Each of them is oriented to a certain class of dynamic systems that could be well, promptly and without obstacles described is it. During the last 50 years, many tens of simulation languages were designed, correspondingly to many tens of classes of dynamic systems that attracted attention of the simulationists.

The translation of the text in a simulation language into the corresponding computer code is a very knotty process, as it should well handle all texts in the simulation language. Implementing it in a form of a compiler is a difficult challenge even for the high level programmers; they are not at disposal, while new classes of systems, which demand their own simulation languages, arise. Already in the sixties of the XX century, this situation stimulated the elaboration of a special technique of representation of concepts, carried by programming language SIMULA [3]-[5].

Many years after, certain ideas existing in SIMULA were followed by the world professional community under title *object-oriented programming* (shortly OOP) and stepwise reflected in other programming languages like SmallTalk, C++, Eiffel, or Java. The ideas were

- (1) representation of general concepts (commonly under name *class*) with
- (2) an absolute freedom to generate any number of their *instances* to represent individuals carrying the contents of the classes, and with
- (3) an absolute freedom to *specialize* such classes to their *subclasses* to represent concepts with a richer content,
- (4) the content of the classes consists in their *attributes* (representing “properties”) and *methods* (algorithms, representing “abilities”), so that
- (5) every instance of a class carries all attributes introduced for the class, and is able to perform any method introduced for the class, when it is demanded to do that, and that
- (6) the specialization of a class consists in adding new attributes and new methods and

(7) giving new contents to methods that have been introduced for the class,

(8) values of attributes two different instances of the same class may differ but the name of the attributes of both the instances must correspond.

The event when an instance N is demanding an instance K to perform method M is commonly called a *message* sent by *sender* N to *addressee* K , having *selector* F . A message can have parameters. The methods the contents of which be changed in the subclasses (see (7)), are called *virtual*. If such a method F is the selector in a message the addressee itself determines the way, according which the reaction to the message will be performed. If F is not virtual then the way is uniquely determined by the sender.

Although the mentioned ideas rooted in the lack of the simulation languages they were already in [3] presented as applicable in programming anywhere. The next development confirmed that so strongly, that some years later the world professional community forgot the simulation stimulus for OOP.

II. LIFE RULES

One of the most important sources of the systems complexity is that they have many elements which behave more or less autonomously and when they get in suitable states they may interact. Viewing the system (and the corresponding model) globally, the sequence of such interactions seems to be chaotic; viewing an individual element, its interactions are recognized as regular; but the stimuli for them, coming from the outside of the element, seem to appear as chaotic.

The authors of the simulation languages were aware of that phenomenon already at the beginning of the sixties of the last century. Into some simulation languages, they incorporated certain ability, later called *life rules*: an element could be connected with an algorithm that could modify its state at one side and react to the states of other elements at the other side. So called *scheduling statements* were at disposal in description of the life rules, enabling switching among life rules of different elements: while the life rules of an element A control the computation belonging to the model their stream could come to a scheduling statement and devolve the control of computation on another element. Later on, an element can devolve the control of the computation on A , which thus “actively” goes on in his life according to the rules that follow after the scheduling statement.

An example of the A 's scheduling statement is “wait until the simulated time increases to a certain value q ”; while the time is increasing life rules of other elements can influence the computing, and when the time accesses q the control of computing is switched to the life rules of A . So a parallel development of more elements can be modeled at a monoprocessor computer; in principle, that modeling is deterministic, therefore can be reproduced.

The first and most popular simulation language that offered the life rules was GPSS [6] (applied even at the present days

namely in the United States [7]). Later, the authors of SIMULA offered to describe life rules for any class C so that

(a) any instance of C should behave according them,

(b) any subclass D of C accepts the rules of C as its owns and an can enrich them by the rules introduced in the formulation of D ,

(c) the scheduling statements were introduced as special cases of *sequencing procedures*, the semantics of which being independent of simulated time.

III. AGENTS

The technique of simulation led to see two sorts of autonomous subjects, namely those recognized in the physical world and those mapping them in simulation models. In [8], term *w-agents* was introduced for the first sort and term *c-agents* for the second one. Simulation practice and model programming demonstrates many structural similarities between the corresponding pairs of a w-agent and the c-agent that models it, among that more or less autonomy appears.

The ideas of OOP, commonly accepted by the world professional community, did not respect the life rules that have existed since the sixties and that have been included into the first OOP language SIMULA. During many years after SIMULA, the following OOP languages (like SmallTalk or C++) offered only formulating some rules applicable during the instance generation. No switching is possible and therefore speaking on *life* rules would be rather illogic – for those languages the life of objects would be a life of ephemeron.

But the parallel dynamics does exist and could be watched, either (outside of simulation) at the real world phenomena, or (independently of simulation) inside the parallel processes of computer systems. Expressing such a dynamic led to the term agent, which could be at hand sometimes as w-agent, sometimes as c-agent. In the scientific literature, agent is not understood in a unique and clear form; some properties of agent are demanded (like autonomous existing and operating, mobility, reactivity, environment understanding etc.), but these properties are not exactly formulated and never classified whether necessary or sufficient.

Let us rest at the opinion that certain autonomy is sufficient to characterize agent. A certain proof of it is [9], appearing in a highly professional conference proceedings oriented to agents in simulation – the model presented in the paper is fully based on the simulation language GPSS mentioned above (see the end of part 2).

IV. SIMULATING AND SUBORDINATED AGENTS

Therefore the c-agents are indeed agents with autonomous dynamics. The essential component of OOP, namely reactions to the messages, causes them to have another ability frequently demanded for the agents, namely *reactivity*; note that that is “more intelligent” in case the selector of a message is virtual. Nevertheless, the experts studying agents introduced term *reactive agent* as an opposite one to the term *intelligent agent*: the first should be much more primitive in

its reactivity than the last. In the next section, we will study the attempts to distinguishing intelligent agent from the poor reactive (i.e. really not intelligent) agents. But for that purpose we need to introduce two concepts, which will be done in this section.

A simulation model itself can be an agent. If it is so we will call it *simulating agent*. The attempt Φ implies that the simulating agents are often viewed as communities of c-agents that form it. Let them be called *subordinated agents*, as they are subordinated to their community. In the terminology related to the agents, such subordinated agents are *delegated* by the corresponding simulating agent.

Fig. 1 presents an illustration. One can see that there is no barrier between the community C and the simulating agent itself, and in practice there is often no difference between the community and the simulating agent.

Sometimes – namely in conventional simulation studies – using the term *simulating agents* can be hammy and functionless. But in some sophisticated simulation studies (e.g. [10]-[12]), the simulating agents figure really as agents, they are autonomous (among other, each of them has its own simulated time) and reactive (better – more or less intelligent), and they form a community (note that they figure as simulation models contemporarily existing in a simulation study).

Note that the concept of subordinated agent can be generalized to w-agent too: the components of any (material) system, which we see as autonomously behaving, can be viewed as w-agents that are subordinated to their community, i.e. to the system S where they exist. A secure test to confirm that such elements are really subordinated agents consists in that such elements could be mapped to agents subordinated to the model of S in case S is simulated.

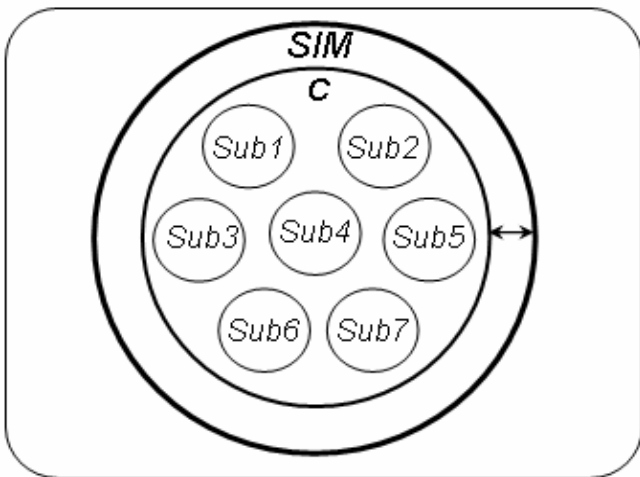


Fig. 1 Simulating agent SIM and community C of 7 subordinated c-agents. They map the w-agents forming a simulated system S . With them, SIM represents a simulation model of S . The short double-side arrow illustrates that the C can be matched with SIM

V. INTELLIGENT SIMULATING AGENTS

As it was mentioned at the previous section, term *intelligent agent* is used as opposing to term reactive agent. It

may be said that intelligent agents perform a certain processing Π of the information that they have got and that they view as essential for the form of the reaction to a message. The adjective *intelligent* should express that Π is supposed to model something near to the mental processes typical for humans. The specialists relate Π either to some learning or to some deduction that could be expressed in the terms of predicate calculus of the first order or in fuzzy logic.

Surprisingly, almost nobody discovered that humans often make something which could be called mental simulation, i.e. something that resembles computer simulation but is performed in a human imagination controlled by reason. One often decides so that he imagines future steps what to do, recognizes the causal relations among them and then “sees in his mental sight” possible consequences, and possibly makes it several times in order to recognize the consequences of different variants, then chooses the variant with the best consequences and decides to do according to it. Naturally the mental abilities of humans are poor for that, and we can watch that it was just simulation (so called “on line one”) that had to replace the mentioned feeble human thinking process by a much more powerful tool.

An element of a real system that behaves in such a manner can belong to a system that should be simulated. As examples, passengers using a system of public transport or walking in a crowded environment, drivers in a road transport, or students transferring from lecture halls into other ones, or even some workers in a production system can serve.

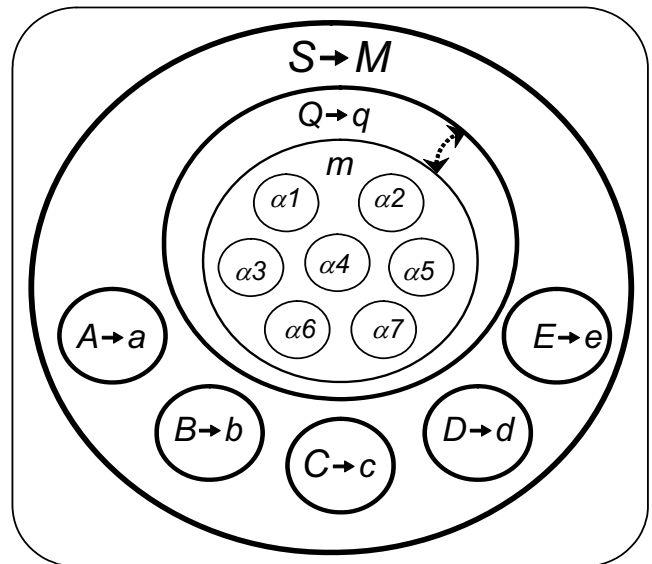


Fig. 2 Simulated system S contains six w-agents $A - E$ and Q . Q can simulate (or imagine), using model m that has seven subordinated agents $\alpha 1 - \alpha 7$. In simulation of S , S melts into simulating agent M , and its elements melt into subordinated agents $a - e$ and q . The boundaries of communities unite with those of models. Concerning the double-sided dotted arrow, see section VI

Such elements are normally subordinated w-agents. When the system in that they exist is simulated they are mapped to subordinated c-agents. But their ability to perform “mental

simulation” gives them a new quality: in fact they are near to be simulating agents and when they are simulated their images – subordinated c-agents – become simulating agents.

In the simulation models of such (real) systems, it is suitable to reflect the mentioned imagining processes as simulation. Moreover, simulation performed in the described manner could inform about the real system behavior in case the human decision would be replaced by computer simulation, e.g. when control of such a system is fully automated.

So we come to the idea to simulate a system S containing at least one simulating element Q that influences S (see Fig. 2). Such an element belongs to the community C of the other elements that form S . In the corresponding model (simulating agent) M , the element Q should be mapped as a subordinated agent q (because it belongs to the community of subordinated agents that reflect the elements of C). At the same time, q should be considered as simulating agent, because it reflects the simulation performed by Q . In [13] and [14] a proof was presented that any neglecting this fact (e.g. neglecting that q is a simulating agent) is a testimony on bad designer’s professional level – either M should give bad information on the future behavior of S , or simulation performed by Q is useless and can be completely removed or replaced by a simpler method even in S .

Let the agents that are simulating and subordinated as well, be called **composed agents**.

VI. IMPLEMENTATION OF COMPOSED AGENTS

No reason exists against applying the attempt Φ in case a composed agent A occurs. Describing it, demands a programming tool available, in which it would be possible to describe both

- (a) the aspects that make A subordinated, and
- (b) the aspect that make A simulating.

To be exact, let us note that the description concerns a w-agent (i.e. a physical computer or a real imagining person) and is expected to be considered as the description of the corresponding c-agent, i.e. to the image of A at computer model M .

The aspects sub (a) concern the fact that A exists in a certain physical world viewed as a system S in Newtonian time, and that in the same world other components of S exist and can interact with A . If A were not simulating, a simulation language could enable such a description; in case no suitable simulation language were at disposal, it would be possible to define it, formulating the necessary language tools (classes, methods,...) in an OOP language. Naturally, the OOP languages like Pascal or C++, which do not allow formulating life rules, could make problems but it is not the objective of the present paper.

The aspects sub (b) concern the fact that A carries (or even is – see the double-sided dotted arrow in Fig. 2) a simulation model m . Relating to S , A has some electronic or neural (or mental) components that exist in S in the same manner as A itself, but A makes an abstraction α , namely a sophisticated

transformation with them, seeing them as carriers of a certain abstract phenomenon, namely a model m that reflects some system S' which is more or less similar to S . But there is no physical relation between the components of S and those of S' – if one could formulate such a relation he had to base it on the abstraction α (among other, m should be bound to a certain fictive time axis that should be different from that figuring for M). Evidently, any modeler would like to neglect any analysis of such a relation – it is physically based on the internal function of the applied compiler which would surely be very complicated. (in other words, to analyze how the abstraction roots in the physics of S would be the same as analyzing the internal details of M inside the computer – it is against the attempt Φ).

If A were not a subordinated agent and if it were only a simulating one, a suitable simulation language could be applied for describing S' or – in case of a lack of such a language – an OOP language could aid to formulate such a simulation language. But the reality is as follows:

At one hand, A is a subordinated agent with its life rules related to the time flow of S , while at the other hand, among the life rules there is a “simulation phase” of the life, in which the description δ of S' should occur. To extract δ from the life rules of A would be against the principles of Φ , as S' is always influenced by the instantaneous state of A .

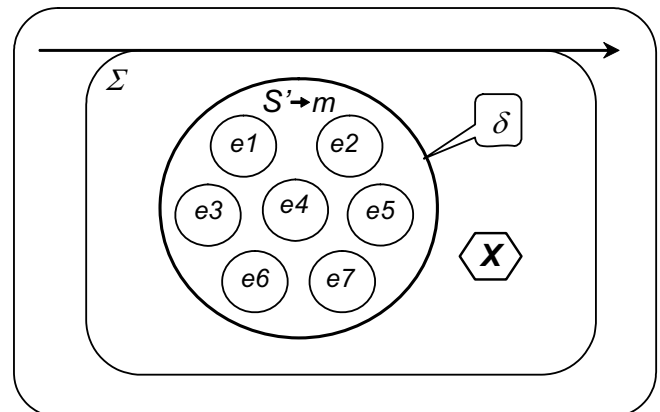


Fig. 3 Q 's life rule(s) Σ containing description δ of S' . For illustration, an entity X accessible (and/or manipulated) by Σ is presented; it can be used in δ .

No existing simulation language permits to describe such a situation (two time axes are sufficient for making the language unable) and similar problems arise in applying OOP. In [8] the essential principles of a user-friendly describing of such composed agents are presented, having use of the OOP that allow formulating life rules and that are block-oriented, too: a consequence of the block orientation is the permission to formulate classes local in blocks or in other classes. System S' can be then described as local to some set Σ of the life rules and so its model can be generated by them and the model itself can communicate with them. Σ can contain any life rules, i.e. from one of them to all life rules of Q . (see Fig. 3).

There is a very small number of programming languages

that have all the three orientations; beside SIMULA, it is BETA and may be JAVA.

VII. CLASSIFICATION OF AGENTS IN SIMULATION

A simple summarizing of the categories of agents occurring in simulation is in Fig. 4. Nevertheless, one can go further. Let a model handled by a composed agent be called *nested model*.

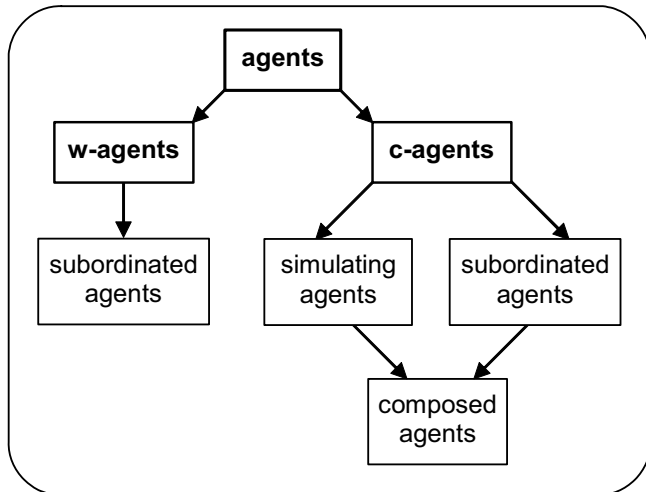


Fig. 4 Simple classification of agents occurring in simulation

At first, note that an agent can be sometimes simulating and sometimes no. That is important for the subordinated agents, which can operate during some time without any model they could handle and which can change such models. Such an agent may correspond to a simulating computer that is sometimes used for other tasks than simulation and that can change its nested models. Such an agent may also reflect a human who imagines only sometimes, his imagining concerns different thinks and situations, while in the remaining time she does not imagine. Note that almost all imagining persons and all simulating computers are of that sort. Therefore a subordinated agent can sometimes be composed and sometimes not, the concept of composed agent is dynamic.

Into one's consideration on simulation, it is possible to include a w-agent which is a computer that performs two or more simulation in multitasking mode (see Fig. 5). The corresponding c-agent should be a composed one which can handle several simulation models at the same time. Although such an idea could seem rather theoretical, SIMULA allows implementing it and a certain application of it exists [15].

That leads to introduce the aspect of subordinated agents, which will be called *size*. It is defined as the number of models handled by the same agent. Therefore in the preceding section the composed agent were those of *size=1*. Naturally, one could generalize the concept of size and view the simulating agents that are not composed as composed agents of size 0; nevertheless such an idea could be a poor mathematical l'art-pour-l'artism. But size is a dynamic value and we can have use of that to admit it for any subordinated agent so that it can vary from zero to a certain maximal value, which informs of the agent's *power*.

Let A be a composed agent subordinated to a simulating agent M that simulates a system S . Let μ be a model nested in A . μ simulates a system S' that can be rather similar to S . Then A be called *reflective agent*. That seems to be a habitual case, while that of a rather great difference between S and S' could seem a theoretical abstraction. Nevertheless, such cases exist, concerning so called *fictive simulation* or *pseudosimulation*: A could simulate a fictitious system in order to solve some particular problem. The first applied examples is described in [16], a certain early collection is presented in [17].

Whether an agent is reflective or not, is a problematic question in general, because the words "S' is rather similar to S" are too fuzzy. Fortunately, the reason of the application of the nested model can help. But there is another complication: an agent of a power greater than two can be reflective in relation to one of the models nested in it and not reflective in relation to another agent nested in it. Such a case was applied and described in [18], where one of the models (far from causing reflectivity) simulates a fictitious system for computing the shortest path in a labyrinth and the other model tests whether using the computed path will not will not cause a deadlock; the last model leads to reflectivity.

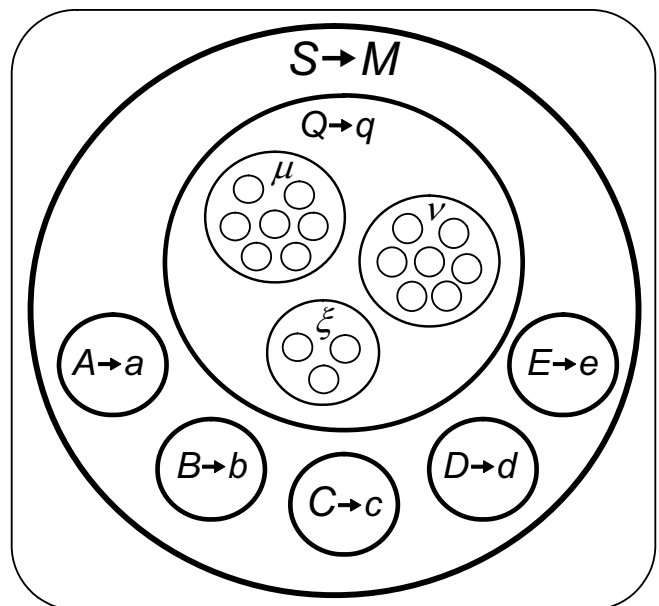


Fig. 5 A similar system S like in Fig. 2, similarly modeled by M ; but computer Q handles three models μ , ν and ξ contemporarily

Finally, there is another occasion to classify the composed agents, which could be called *depth*. A subordinated agent of power zero can be classified as power of *depth zero*: a model is never nested in it. Suppose a simulating agent M is nested in a subordinated agent A and suppose among the agents subordinated to M is an agent B that is composed. Such a situation corresponds to a system S that has element Q that simulates a system S' containing an element q that is simulating, too (note that an experimental model of S was already implemented and run [19]). In such a case, A be called agent of *depth two* (see Fig. 6). In the "more habitual" case,

i.e. if M has no composed subordinated agents, A be classified as that of *depth one*.

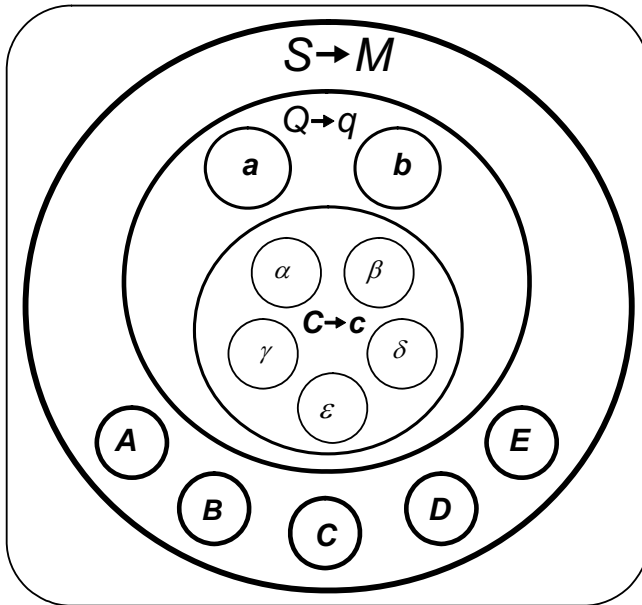


Fig. 6 System S like in Fig. 2, similarly modeled by M ; but computer Q models a system composed of three elements a , b and c , and c reflects a computer C that simulates a system composed of five elements $\alpha - \epsilon$

Evidently, the nesting can be iterated so that q simulates a system containing a composed agent etc., and so we can come to the depth three etc. Evidently, for an agent A of power two or more, the depth can differ when it is watched in relation to different agents subordinated to A . Therefore depth could be in general expressed by tree of subordination.

VIII. CONCLUSION

Nowadays, it is necessary to test the simulation of anticipatory simulating systems as much as possible, in order to get new stimuli and to generalize what was already reached. The further application studies concern e.g. the hospitals [20-22] and the internal logistics in production halls considered as anticipatory systems [23]. Other work that is prepared to follow the classification is joining it with some taxonomy introduced for the anticipatory systems independently of agents [24]. Also French system QNOP, which appeared a good tool for simulation of production systems [25], is studied to be transformed for to serve for describing composed agents.

ACKNOWLEDGEMENT

The paper was elaborated under support of the grant No. 201/06/0612 of the Grant agency of Czech Republic.

REFERENCES

- [1] O.-J. Dahl: *Discrete Event Simulation Languages*. Oslo: Norsk Regnesentralen, 1966. Reprinted in [2]
- [2] F. Genuys, Ed.: *Programming Languages*. London – New York: Academic Press, 1968
- [3] O.-J. Dahl and K. Nygaard: "Class and subclass declarations", in *Simulation Programming Languages*, J. N. Buxton, Ed. Amsterdam: North-Holland, 1968, pp. 158-174
- [4] O.-J. Dahl, B. Myrhaug and K. Nygaard: *Common Base Language*. Oslo: Norsk Regnesentralen, 1968 (1st ed.). 1972 (2nd ed.), 1982 (3rd ed.), 1984 (4th ed.)
- [5] *SIMULA Standard as Defined by the SIMULA Standards Group*. Oslo: Simula a.s., 1989R. Rosen: *Anticipatory Systems*. New York: Pergamon Press, 1985
- [6] G. Gordon: "A general purpose systems simulation program". *Proceeding 1961 EJCC*, New York: MacMillan, pp. 81-88
- [7] T. J. Schriber: *An Introduction to Simulation using GPSS/H*, New York: Wiley, 1991
- [8] E. Kindler: "Object-Oriented Simulation of Simulating Anticipatory Systems". *International Journal of Computer Science*, vol. 1., no. 3, pp. 163-171, 2006
- [9] E. Kalisz and A. M. Florea: "A GPSS Simulation model of Interactions in a Market –Based Multi-Agent System" in *Workshop 2000 Agent-Based Simulation*, B. Schmidt, Ed. Delft, Erlangen, Ghernt, San Diego: CSC & ASIM, 2000, pp. 145-190
- [10] J. Weinberger and E. Kindler: "Experimenting in quasiparallel". *Simula Newsletter*, vol. 10, p. 12, 1982
- [11] J. Weinberger: "Extremization of Vector Criteria of Simulation Models by Means of Quasi-Parallel Handling". *Computers and Artificial Intelligence*, vol. 3, pp. 71-79, 1987.
- [12] J. Weinberger: "Evolutional Approach to Extremization of Vector Criteria of Simulation Models". *Acta Universitatis Carolinae Medica*, vol. 34, pp. 249-258, 1988
- [13] E. Kindler: "Chance for Simula", in *Proceedings of the 25th Conference of the ASU – System Modelling Using Object-Oriented Simulation and Analysis*. Kisten (Sweden): ASU, 1999, pp. 29-53. Reprinted as [14]
- [14] E. Kindler: "Chance for SIMULA". *ASU Newsletter*, vol. 26, no. 1, May 2000, pp. 2-26. Reprint of [13]
- [15] E. Kindler: "When everybody anticipates in a different way ...", in *Computing Anticipatory Systems CASYS 2001 – Fifth International Conference*, D. M. Dubois, Ed. Melville, New York: American Institute of Physics, 2002, pp. 119-127
- [16] E. Kindler and M. Brejcha: "An application of main class nesting – Lee's algorithm". *SIMULA Newsletter*, vol. 13, no.3, pp. 24-26, 1990
- [17] E. Kindler: "Simulation of Systems Containing Simulating Elements", in *Modelling and Simulation 1995, Proceedings of the 1995 European Simulation Multiconference*, M. Snorek, M. Sujansky, A. Verbraeck, Eds. San Diego: Society for Computer Simulation International, pp. 609-613, 1995
- [18] E. Kindler: "Nesting simulation of a container terminal operating with its own simulation model". *Belgian Journal of Operations Research, Statistics and Computer Sciences*, vol. 40, no. 3-4, pp. 169-181, Dec. 2000
- [19] P. Blümel and E. Kindler: "Simulation of antagonist mutually simulating systems," in *Simulation und Animation '97*, O. Deussen and P. Lorenz, Eds. Erlangen, Ghent, Budapest, San Diego: Society for Computer Simulation International, 1997, pp. 56-65
- [20] E. Kindler and I. Křivý: "On the way to reflective simulation of hospitals," in *4th International Conference Aplimat, Part II*. Bratislava: Slovak University of Technology, 2005, pp. 309-314
- [21] I. Křivý, E. Kindler: "Computer representation of formalized view of in-patient departments of hospitals" in *CompSysTech 2005 – Proceedings of the International Conference on Computer Systems and Technologies*, Varna: Bulgarian Union of Automation and Informatics, Varna, 2005, pp. 1-6
- [22] I. Křivý, E. Kindler: "Reflective Simulation of In-Patients Dynamics", in: *5th International Conference APLIMAT 2006*, Part I, M. Kováčová, Ed. Bratislava: Slovak University of Technology, 2006, pp. 613-617
- [23] E. Kindler, T. Coudert and P. Berruet: "Component-based simulation for a reconfiguration study of transitive systems", *SIMULATION*, vol. 80, no. 3, pp.153-163, March 2004
- [24] E. Kindler, I. Křivý and A. Tanguy: "Object-oriented system analysis of anticipatory systems in week sense". *International Journal of Computing Anticipatory Systems*, vol. 14, pp. 271-285, 2004
- [25] A. Tanguy: Implementation and application of a modelling environment for manufacturing systems. In: *Application of Distributed & Graphical Simulation*, Aberdeen (Scotland): King's College, Aberdeen, UK, pp. B-2-1 – B-2-12, 1993