University of Ostrava

Institute for Research and Applications of Fuzzy Modeling

Object-Oriented Simulation of Simulating Anticipatory Systems

Evžen Kindler

Research report No. 97

2006

Submitted/to appear:

ICCS (International Conference on Computing Science), Prague, 2006

Supported by:

Institutional research scheme MSM6198898701 of the Czech Ministry of Education, Youth and Sport

University of Ostrava
Institute for Research and Applications of Fuzzy Modeling
30. dubna 22, 701 03 Ostrava 1, Czech Republic

tel.: +420-59-6160234 fax: +420-59-6120 478
e-mail: evzen.kindler@osu.cz

Object-Oriented Simulation of Simulating Anticipatory Systems

Eugene Kindler

Abstract — The present paper is oriented to problems of simulation of anticipatory systems, namely those that use simulation models for the aid of anticipation. A certain analogy between use of simulation and imagining will be applied to make the explication more comprehensible. The paper will be completed by notes of problems and by some existing applications. The problems consist in the fact that simulation of the mentioned anticipatory systems end is simulation of simulating systems, i.e. in computer models handling two or more modeled time axes that should be mapped to real time flow in a non-descent manner. Languages oriented to objects, processes and blocks can be used to surmount the problems.

Keywords — Anticipatory systems, Nested computer models, Discrete event simulation, Simula.

I. Imagination and Simulation in Design

WHEN a collective of humans (a team etc., exceptionally one human) *S* designs, develops, organizes and/or constructs a system *s*, almost always such an activity is performed with respect to future: *S* assumes *s* not only to be once made as a static thing, but frequently takes into account that *s* will behave and exist during time in order to satisfy some task, target, intention etc. *S* anticipates the possible behavior of *s* by using of a certain model *M* and – according to that anticipation – *S* frequently changes the conception of the future *s*. According to the definition introduced in [1], *S* is an ***anticipatory system***, because the instantaneous states of its work are influenced by viewing to the future (to possible behavior of the result *s* of its design work) and that viewing may lead *S* to change the reactions to its instantaneous states.

M is a model of *s* and it often a mental one, frequently aided by imagining in sense "What would *s* do after it is realized and applied as a real object in the physical world?" The capacity of human imagining is rather limited, humans make often errors, the imagining should be controlled by rationality, and therefore computer simulation is often applied to replace human imagination, namely in case *s* is rather complex. In general, computer modeling permits to take a great (may be variable) number of (often variable) properties into account, and computer simulation permits to add causality of events into their variability. The simulation model *M* is carried by a certain computer *C* that can be viewed as an element of *S* (see Fig. 1 where some possible elements of *S* are denoted as *H1-H6*).

To program a simulation model of a complex system is a difficult task and therefore ***simulation languages*** were developed; their basic contribution is that their user does not need describe what should happen in the simulating computer, but

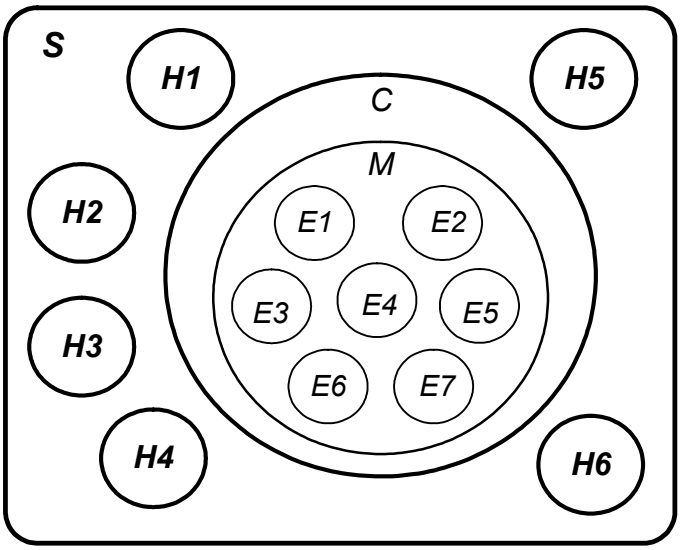

Fig. 1 Designing team composed of 6 humans *H1-H6* and computer *C* that contains model *M* of *s*, composed of 7 elements *E1-E7*

he only describes the system that should be simulated; then the description can be automatically interpreted or translated into a computer program. More and more complex systems demand to be simulated, the developing of simulation language processors was hard and expensive and therefore already in 1966 the principles of the object-oriented programming (further OOP) was designed and in a short time after, in Simula programming language offered to the users. In OOP, the classes enable representing general concepts, dynamic creating of their instances enables to form models with time dependent structures, the subclasses enable effective organizing the concepts according to their content and extent, the messages enable to introduce a certain standardization of the abilities of instances to interact either effectively or – in case of virtual methods – in a symbolic way that can be completed or even re-declared when a concept of richer contents is defined. OOP enables to define problem-oriented programming languages without hard and expensive

Manuscript received February 10, 2006.

Eugene Kindler is with the Institute of fuzzy modelling and applications, Faculty of Science, University of Ostrava, 30. dubna Street no. 22. CZ – 70103 Ostrava, Czech Republic (phone: +420-221-914-286; fax: +420-221-914-323; e-mail: evzen.kindler@ mff.cuni.cz).

implementation of their processors: a class *D* of (possible) real systems is analyzed, every partial result of the analyze can be immediately recorded in an OOP language and when the analyze is finished the text in the language, got during it represents a definition of a problem-oriented language, prepared to model the systems of *D*. Among such problem-oriented languages, new simulation one arise, "tailored" to various classes of (complex) systems.

II. Imagination and Simulation in Operation

Let us now turn attention to system *s* designed and realized by the system *S* (see section I). Frequently, *s* is supposed to operate under a certain small or less human control or at least influence (note that even a manual worker employed in *s* is a component of *s* and can affect the future states of *s*). Let *h* denotes such a human. He is also an anticipatory system, using often a certain mental model that leads him to make some decisions, which may directly or indirectly influence the future dynamics of *s*.

More humans with their own models can affect the same system. Therefore the automatic control is applied with a vision to concentrate the decision in one center and to replace all mental models by a unique one *m*. Nowadays computer simulation models come increasingly into interest of specialists.

Suppose inside *s* at a certain state an intervention is necessary and a set of possible interventions exists, forming a question to choose the intervention that causes the optimal consequences in a certain future time; to determine the consequences is often difficult and thus for every possible intervention a simulation model can be run to present the future consequences, so that finally the optimum decision is known. The simplest way is to let the models run one by one. Therefore the computer *c* that manipulates them may carry only one such a model *m* at a time (see Fig. 2, which illustrates a certain similarity between *s* and *m* inside *c*, using a correspondence between εi and $-\varepsilon i$; note that *c* is a member of *s* in the same sense like *e1*-*e7*).

III. Agents

The idea of computing agents (further c-agents) roots in its certain analogy in the world – such agents are often taken as models of agents (further w-agents) existing outside computing technique. It is no more known that since 1960 the first symptoms of c-agents have been in the first discrete event simulation language GPSS [2] (applied until the present time in a PC form [3]). It offered easy describing simulation models so that their elements were viewed as simple c-agents existing and operating in the common simulated time according to their "life rules": the corporative behavior of the c-agents was automatically implemented by a scheduling mechanism among life rules of different c-agents. One described the simulation model so that he viewed the simulated system as composed of w-agents, which he described in GPSS and their descriptions were interpreted as if they concern c-agents.

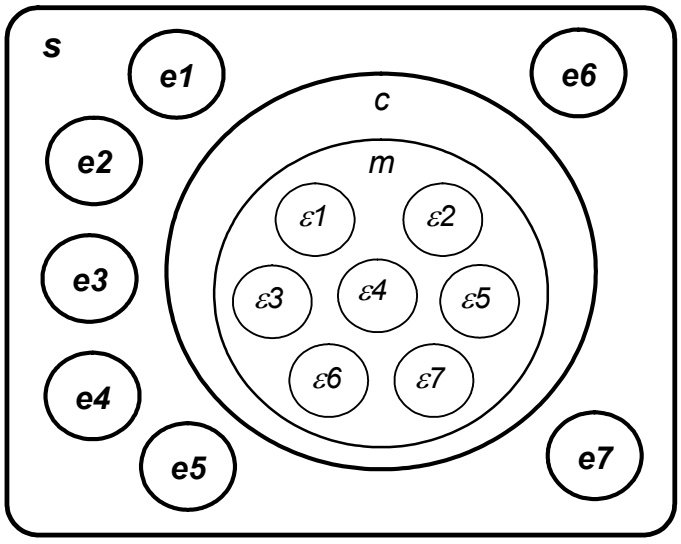

Fig. 2 Designed system composed of 7 elements *e1*-*e7* and of computer *c* that handles model *m* reflecting the environment of *c*

The reactivity and interactivity of such c-agents was rather limited in GPSS, but several years later – still in the sixties of the XX century – other discrete event simulation languages (e.g. SOL or Simula I [4]) admitted very rich interactions. They are called ***process-oriented simulation languages*** [5]. Finally (in 1967), the object-oriented programming (further OOP) was born in the cradle of simulation [7] as a universal programming paradigm; it enabled not only to define classes of c-agents and order them according to their content and extent "homomorphly" with the relations among the corresponding w-agents (the classes of which do not differ from what the philosophers call concepts), but also to introduce common names of their possible interactions. Note that according to [7] and the first implementation Simula 67 [8] of OOP, the "life rules", i.e. the algorithmic description of the c-agent dynamic interruptible behavior remained, while many other OOP programming tools like SmallTalk, C++, Eifel and newer versions of Pascal admit no interruptible "life rules" and their users have difficulties when wishing to formulate c-agents in them.

Nowadays, Simula 67 is officially called simply Simula [9], as it completely covered the use of Simula I. Among the other OOP languages that admit the "life rules" are Beta [10], Java and Modsim [11]. They are both OOP languages and process-oriented ones. It is possible to state that a language that has the both orientations is also ***agent-oriented***.

IV. Simulating and Subordinated Agents

What was described above was oriented to w-agents that form systems (i.e. that are elements of systems intended to be simulated) and the corresponding c-agents that form simulation models. Examples of such w-agents are *H1*-*H6* and *C* of Fig. 1 and *e1*-*e7* and *c* of Fig. 2, examples of such c-agents are *E1*-*E7* of Fig. 1 and $\varepsilon 1$-$\varepsilon 7$ of Fig. 2. Let them be called ***subordinated agents***, as they are components of either

simulated systems (in case they are w-agents) or simulation models (in case they are c-agents).

C and *c* are special ones: they carry simulation models. Let them be called ***simulating agents***. Until the present phase of the explication, it seems that every simulating agent is subordinated w-agent and cannot be subordinated c-agent. The future phase of explication will show that the present idea is not complete.

V. Nesting Simulating Agents

Using usual terms of computing science and system science, we can say that the subordinated w-agents are nested in the system where they belong and the subordinated c-agents are nested in the models where they belong. Note the relation of nesting is often considered transitive (not only expressing a direct nesting). In case the models themselves are considered as agents (see section XI), the transitivity enables to preserve the mentioned nesting of agents also in case the subordinated ones are viewed as nested in models.

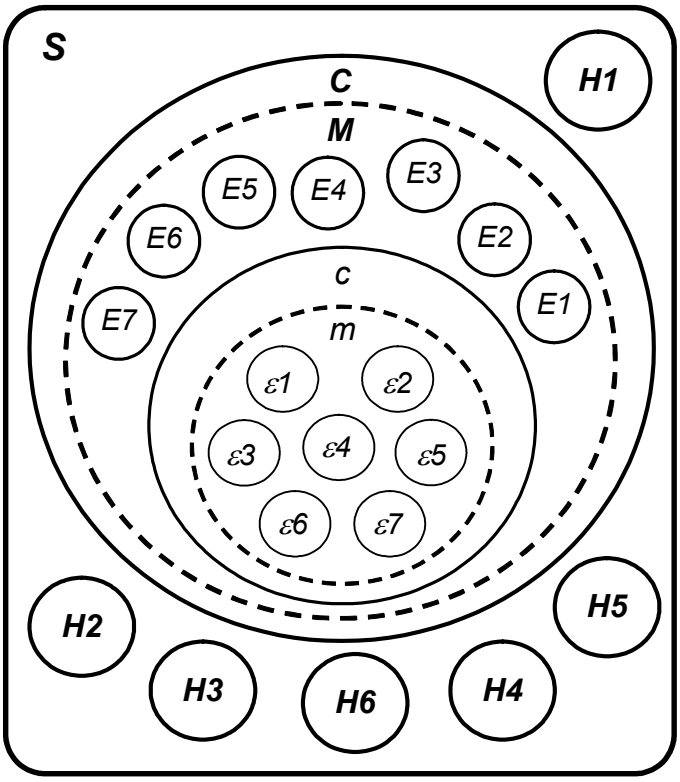

Fig. 3 Nesting of model *m* inside an element of model *M* – each of the models is depicted with dashed line and geometrically nested into the image of the element (computer *C*) that carries it (in case of *M*) or into the image of the model of the element (computer c) that carries it.

It is said with a bit humor that (computer) simulation is the worst method to get data for decision – it needs a model that is usually rather complicated and hard for constructing, and the run of the model needs a lot of computing time. Although the present computers are so fast that the computing time problem is almost negligible (it concerns the personal computers, too), it is true that in case simulation could be replaced by a short formula it would be better. Nevertheless the human civilization meets more and more complex systems, for which simulation is the only method to tell something exact about, while the desire to apply simple formulas is like fata morgana.

Suppose the team *S* designing a system *s* is apprized of its great complexity and of the consequence that in *s* simulation will be sometimes necessary. Then it is necessary to accept such a simulation as a component of the model *M* used by *S* during the design. If *S* is really convinced that *s* will use simulation then the deletion of this simulation from *M* (or its replacing by something simple) would damage *M* and its predicting ability.

The consequence is that model *M* used by *s* must reflect also computer *c* existing in the designed system *s* and the models like *m* handled by *c*. In a graphic scheme, we must accept that the image of *M* in Fig. 1should contain – beside the elements *E1-E7* – also something like the image of *c* presented in Fig. 2. In other words, the contents of *s* visible in Fig.2 should be transferred into the image of *M* visible in Fig. 1 so that the w-agents *e1-e7* change to c-agents *E1-E7*. See Fig. 3.

So we our consideration comes to nesting simulating agents. They correspond to nesting anticipation: the designing system *S* anticipates the behavior of the designed system *s* under the expectation that *s* itself will be anticipatory. Let conventional simulation models be called (simulation) ***models of order one*** and models like *M* be called (simulation) ***models of order two*.**

VI. Problems of Models of Order 2 – More Time Axes

A very profitable property of the simulation languages are tools for automatic manipulation with simulated Newtonian time axis that corresponds to the real time flow viewed at the simulated time; that allows that the events coming in the real time described by means of the tools can be "homomorphly" scheduled in the corresponding simulation model. As there is only one time axis visible as real time flow, the simulation tools offer only one simulated time axis in any simulation experiment. And the standard simulation tools introduced into OOP languages are limited in the same way. But one problem of nesting agents is that they behave in two (or even more) different time axes: a model *M* of order two itself changes its states in a certain time axis *T* while if it calls a model *m* nested in it another time axis *t* arises and exists during the simulation experiment with *m*; then *t* disappears but when another use of this (or another) nested model is demanded, a new time axis *t'* should arise.

As an example concerning the difference between *T* and *t*, the following statement can be presented: "while computer *C* operates during time interval $<T_1, T_2>$ it simulates what could happen in *s* during time interval $<t_1, t_2>$". The only demands for the values are that $T_1 < T_2$ and $t_1 \leq t_2$ but otherwise no limitation exists (really, $<T_1, T_2>$ is often like <56 ms, 62 ms> while $<t_1, t_2>$ may be like <20 days, 23 days>).

The events concerning *M* have to be non-decreasingly orde-

red with respect to *T* according to their times during the whole existence of *M* and in that order they must be interpreted at *C* during real time flow, and that holds also for the events related to *t* and to *t'*; but there are no similar relations concerning the events belonging to different time axes. For example, after finishing a nested experiment by an event scheduled at time axis *t* for rather great time, the time axis *t'* could start with an event scheduled for zero.

VII. More Time Axes – Answer

The existing simulation programming tools appear useless for implementation of nesting models of order two. But there is a good way to implement them, namely by means of the OOP languages that are process-oriented (agent-oriented – see section III) and ***block-oriented***.

Block orientation was exactly introduced into programming in ALGOL 60 language at the beginning of the 60-ies of the preceding century [12] but the enthusiasts of structured and modular programming proscribed it already in the 70-ies. Nowadays, such mode of programming is out of use for a long time but the return of block-orientation is rather reluctant. Simula has had that orientation already since 1967 and then only Beta and Java were equipped with it.

Block can be described as a part of life rules that can manipulate with some "private" entities, i.e. with something that is not accessible from the outside of the block. The entities are called ***local*** in the block and can be variables and subroutines; nevertheless, if a block-oriented language is also OOP one, classes can be among the local entities. When an instance *E* of a class performs its life rules and enters into such a block it can be viewed as representing something that just entered into a certain phase of its life that is "intellectually" richer: *E* behaves as "knowing" what mean the local entities and is "thinking" by means of them. In case such a local entity is a class, the words "knowing" and "thinking" do not sound bombastically, as the instance can use the class for many purposes, among which there is generating instances and letting them perform according to their life rules.

Thus, if an instance *E* enters into a block *B* where classes *U*, *V*, *W*, … are introduced as local ones, *E* can use them to form a model *m*, which behaves like a "private model" existing only in the *E*'s existence. When simulation tools are introduced into the block *B* too, this block behaves as being a simulation experiment and *E* behaves as a simulating agent as far as its dynamics is inside *B*. Let such a block be called ***simulation block***. When leaving *B*, the simulation model *m* disappears together with all what is local in *B*.

In case *E* was a subordinated agent already before entering *B* its events are bound to some time axis *T* related to a model *M*, which *E* forms a component of. The ability to be a subordinated agent is preserved during the *E* is a simulating agent and persists after *E* leaves *B*. Thus *E* can be contemporarily simulating and subordinated. Operating inside *B*, a time axis *t* local to *B* arise and the entities of the classes local to *B* are related to it. A very fine and sophisticated use of Simula tools enables that during operating inside *B*, *E* behaves like a part of the model *m* for that it is a carrier.

VIII. Formal Similarity of Nesting Simulation Models

Habitually, the models *M* and *m* mentioned above concern the same thing, i.e. they should have many similar properties. Let such a phenomenon be called ***reflective simulation***. Nevertheless, both the models can be declared similar only when they are viewed in a formal manner and without their context. Certain parts of their semantics differ: while *M* is a model of *s*, *m* nested inside *M* does be a model of certain electronic phenomena existing in the computer *c* represented in *M*, namely of the phenomena that are viewed as a model of *s*. Logically, the elements of both the models cannot be mixed. E.g. an image of an element in *m* cannot be inserted into a an image of a queue in *M* or vice versa; such a step would model the inserting of a real element of *s* into a representation of a queue inside the electronics of *c* or – in the opposite case – inserting an electronic phenomenon inside *c* viewed as a representation of an element of *s* into real queue existing in *s*. In general, such assignment is a (programming) error called ***transplantation*** and can lead to logical contradictions [13], sometimes interpreted as a collapse of computer or task. Therefore if such an error is made and then recognized according to its consequence (collapse) its repairing is almost impossible.

The best way to be safe against transplantation is to apply rather different languages for describing *M* and *m*, i.e. to give all the entities (the classes, their attributes and methods and their instances) names differing from those introduced in the simulation block corresponding to model *m* among the life rules of *c*. Unfortunately, this idea is very inconvenient, as if accepting it the author of a model of order two would have to remember two languages and use them adequately, contrary to the fact that he describes the same object in them.

The other way would be to give a wide berth from writing on both the models in the same sentences. Unfortunately, it is also impossible, because at least during the start of the nested model *m* its carrier should assign its initial values by those read as the corresponding instantaneous values of *M*.

Therefore the practice demands using the same languages for the description of both *M* and *m* and applying them in sentences where both the models figure. Therefore transplantation is a real danger. It seems that Java checks it sometimes while in other cases the security depends on the user's responsibility. Beta seems to have checks on the security but many of them exert during the run of the compiled model, i.e. lengthen the simulation experiments.

IX. Case of Simula

Simula was designed to check the possible transplantation cases during the compilation as much as possible. Because of it, the following two limitations have to be respected:

a. the blocks are not objects and they cannot get names,

b. the excellent Simula standard tools for simulation (name-

ly for automatic event scheduling) can be applied only to instances of classes local in blocks.

The consequence is that a model cannot be identified and therefore one cannot use assignment statements like

m.machine3.object.mass:=M.machine3.object.mass

telling something like "in the newly generated model *m*, the *mass* of the *object* that is being elaborated at *machine3* should be copied from the corresponding structure existing just in model *M*".

Similarly, the same limitations seem not enabling to distinguish two "scheduling" statements expressing that something takes a certain (simulated) time, for which Simula offers a standard tool *hold(e)* in case of models of order one (if such a statement is met during the performing of life rules of a certain agent *p* it tells *p* to interrupt the performing until the simulated time increases of *e*. To understand the obstacle, one must realize that the carrier *c* of model *m* is contemporarily a subordinated agent in *M* and a simulating agent carrying *m* and that *hold(e)* could be demanded to relate to any of time axis *T* or *t*. The former interpretation tells that inside *S*, the computer *c* should wait *e* units, while the latter interpretation tells that the subordinated agent *a* that just controls the computation inside *m* should wait *e* units. In other words, the first interpretation tells something on the computing rate of *c* while the second interpretation speaks on what happens inside *m*. Simula does not allow to distinguish both the cases in a simple way like *M.hold(e)* or *c.hold(e)* and *m.hold(e)* or *a.hold(e)*; it allows to write only *hold(e)* and to interpret it as *a.hold(e)*.

X. First Solutions of Problems

It is necessary to say that for a scientific analysis a modern programming language that is object-oriented, process-oriented and block-oriented represents something like formal mathematical theory; the description of the languages is like the axioms and the application of the language is like deriving theorems from the axioms. The axioms of certain mathematical theories were stimuli for such deriving during hundreds of years and a similar situation can be met at the modern programming languages. Note that only the syntax of such languages is expressed by more that hundred rules that behave as axioms that cannot be generated by some algorithm, and that the languages that are object-oriented and block-oriented behave like theories of systems, the elements of which can be other theories.

Respecting such a situation, there is not a surprise that 25 years starting by the exact definition of Simula a common opinion existed that Simula is so secure against transplantation that it does not allow any communication among simulation models (i.e. in case each of them had its own time axis, independent of those of the other models). But in 1993 a trick was discovered how to circumvent its restrictions and a way to make possible the communication among different simulation models and thus the real use of reflective simulation, too [14].

The trick was sufficient but rather sophisticated and not suitable for wholesale application. After having implementing some models (see section XII) an idea came to prepare software that could help the users to surmount the captiousness of the trick: the objective of the idea was to make a processor that would translate a Simula description of a model of order one to a description of a similar model enriched by an image of a computer that would be able to react to a certain message so that it watches its environment, creates a model of it and carries that model as a block of its own life rules. When a user would let send such a message he could be sure the computer to have the model and the users would only complete a description by the criteria, for which he desires the nested model.

The software was implemented as a structure of three programs in Simula – the processor mentioned above, the run time support for the run of the translated (and completed) models and a program for checking whether the source of the processsor has not errors. It worked in a satisfying manner [16], but in 2005 a rather revolutionary fact was discovered on Simula.

XI. Models as Agents

In order to explain the discovery let us illustrate the preceding situation in the models of order two in a graphical manner,

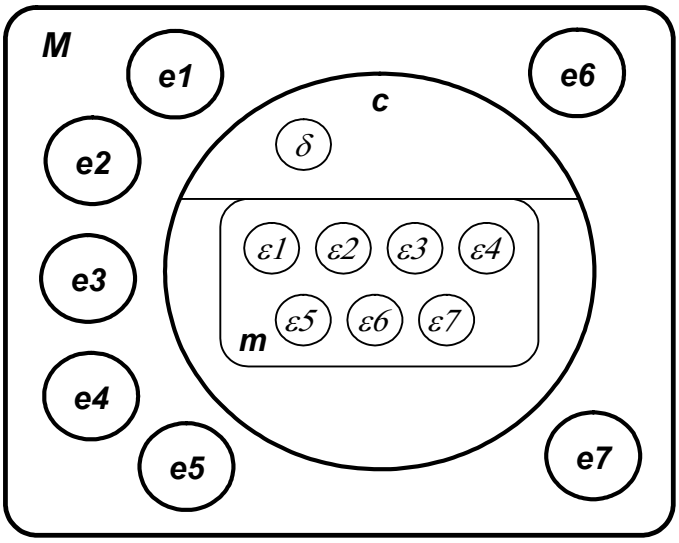

Fig. 4 Implementation of model *m* carried by a simulating agent *c*, having use of the standard simulation tool offered by Simula

similarly to the preceding Figures (so called Mejtsky's diagrams were used, which enable understanding the nesting classes, models and theories [17], [18]).

Fig. 4 corresponds to Fig. 2 but respects the restriction demanded by Simula: the horizontal abscissa inside the image of *c* represents the dynamics of *c*. When it enters in a simulation block, the block is "attached" to the dynamics and a model *m* arises inside it. Blocks are depicted as rectangles with rounded vertices. Simula allows giving names to the objects, i.e. to the entities that are depicted as circles, but not to blocks, i.e. to the objects depicted in other form. Let "dot notation" of form *A.B*

represents "*B* of *A*"; then it should denote something like a penetrating into inside of the circle *A* and to point there an object called *B*. The boundary of a rectangle is closed for penetrating. Therefore e.g. *e2* can manipulate $c.\delta$ but not $c.\varepsilon 2$, and δ – being outside *m* – cannot be a subordinated agent in *m*.

Suppose Fig. 4 represents a model *M* applied in the design. For the visual understanding, in the Figure, *e1* and $\varepsilon 1$ differ by names but what happens when they have the same names in a Simula program (e.g. *name_k*, for *k*=1, …, 7)? Then the elements like *e1*,…, *e7*and *c* cannot penetrate to the block *m* inside *c* and thus to communicate with $\varepsilon 1$,…, $\varepsilon 7$; but also $\varepsilon 1$,…, $\varepsilon 7$ cannot communicate with *e1*,…, *e7*, because *ek* and εk (*k*=1, …,7) have the same names *name_k* that cannot be distinguished e.g. by dot notation (neither *M* nor *m* have names) and – according to Simula rules – *name_k* occurring outside the block of *m* denotes only *ek* while *name_k* inside this block denotes only εk.

The innovation of 2005 consists in using another simulation tool than that offered as standard one by Simula. A new tool was programmed so that it does not demand place the simulation models into blocks: such models can figure as agents, each of them can get a name and using that name they can communicate. What was thus depicted in Fig. 4 was returned to the situation depicted in Fig. 2 and the whole model of order two is organized exactly as depicted in Fig. 3. For example, model *M* can communicate by an element εk of *m* by means of *c.m.N* where *N* is the name of εk. Similarly, *m* can communicate by an element of *M*, which carries the same name *N*, by using or even *M.N*.

Note the way to the result took 38 years of "thinking in Simula" and that the further details would demand very deep knowledge of this language. Nevertheless it should be mentioned that the old rules of Simula cause the new simulation tool to be completely secure against the transplantation.

XII. Applications

In the conclusion of [5] it was demonstrated that the simulation languages can make easy programming of some routines that are far from simulation in case they are viewed from purely physical viewpoint. When a computing process *A* is necessary to be programmed, then – instead of algorithmizing it – one can imagine a fictitious system *F* that generates the same data as *A*, and implement a simulation model of *F*. One speaks on ***fictitious simulation***. We used often Simula for it and since 1990 the fictitious simulation has been nested in "true" simulation. Naturally, the activity was far from the reflective simulation, as the fictitious systems were rather different from those in they were nested. Among the examples presented in [5] a method Φ for computing the short path occurred, applying simulation of fictitious system *F* of multiplying pulses.

The first study [19] concerned a production system served by automatically guided carriages; to get the optimal paths, the simulated carriages applied the mentioned method Φ. The same technique was then applied in preparing software for simulation of automated container yards – the ground-moving tools for the operational transport inside the yards used Φ to get the optimal path composed of free places in the labyrinth columns of containers ([20], pp. 265-274).

An extensive experimentation with the models of container yard showed that almost all cases finished by a deadlock; the sufficient condition was occurring two or more transport tools in the simulated system. The deadlock was caused by the possibility of a change of the configuration of free places at the yard during using a result of Φ: this method was based at the instantaneous configuration of free places at the moment of the computing, but when a transport tool was applying the result of Φ another transport tool could change the configuration and cause a barrier on the computed path. Simulation showed that sooner or later the cumulation of barriers led to total collapse of transport in the yard.

Therefore the model was completed with another nested model *m*, applied as follows. After the shortest path *p* was computed by Φ, model *m* was applied to simulate what consequences can come when *p* is used. If *m* told that no barrier can be expected *p* was applied, and when *m* discovered a barrier at some place *g* of *p*, a fictitious container *f* was placed at *g* and method Φ was newly applied. Because of *f*, the newly computed path was different from *p*, it was tested by *m* etc., until a safe path was reached [21].

Model *m* clearly anticipates a possible future development of the system in that it exists; therefore *m* was the first case of practical application reflective simulation. Before it, a demonstration case of reflective simulation was run, simulating a bank of several tellers and one or more dispatchers who monitor the situation in the queues and possibly can open a new teller (in case it exists and the queues are rather long) or close a teller (in case the queues are short or empty). In order to learn whether such a decision is not a result of a rather volatile situation, each of the dispatchers has a computer at that he simulates the future, in order to see the possible consequences of his decision; according to what the simulation shows, the dispatcher accepts or modifies his decision [22].

Although the mentioned simulation was not directly applied in practice it discovered many aspects, among which models of various types of executives existed like master-pupil or two non-communicating bureaucrats [23] and also simulated pairs of competing systems that use their own simulationists to discover what is the rival's intention or even what the rival's simulationist might simulate about the simulation of the other system [24].

Another application concerned public transport at a region of a Moravian town Havirov of about 100.000 inhabitants: the dynamics of the bus public transport net was simulated so that passengers were represented according to their possible imagining: each of the imagines the duration of possible paths (sequences of walking and using various bus lines) to his target and accordingly he decides for a certain path. The simulation was oriented to future central dispatching and information

center, using portable telephone network to advise the passengers on their paths optimal in relation to the instantaneous situation [25].

The mentioned cases were applied with Simula standard simulation tools, i.e. by hard programming technique depicted in Fig. 4. In the next class of applications, one started with the mentioned difficult technique but nowadays continues with the new simulation tool mentioned at the end of section XI. It concerns circular conveyor with rollers, to which some working areas are connected (see Fig. 5). When such an industrial transport system is designed, the questions relate not only to the number of the working places, to the length of the main

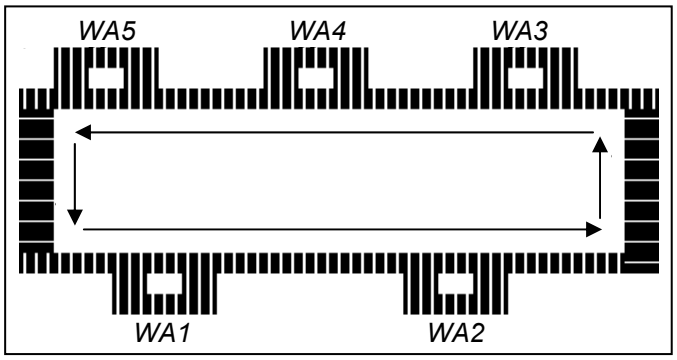

Fig. 5 An example of a conveyor with 5 working areas

circle and to the power of the driving mechanism but also the future control that should eliminate crashes and that will answer questions like the follows:

(1) When an object comes to the conveyor and the conveyor is rather occupied: should the object enter it or not (when it enters, it might be returned after making a whole cycle at the conveyor basic circle without any results and it might obstruct the transport of other objects during that).

(2) In case the object has to wait for being accepted to the conveyor, what is the duration of the waiting?

(3) In case of an unexpected fault: is it better to stop the conveyor function immediately and to repair the fault, or to continue some time with a limited number of working areas?

(4) When the decision in (3) is to continue, what modification of the technological programs would be optimal?

Note the question (4) is rather general and falls into many other questions. All should be answered by simulation during the conveyor operation and each of such simulation applications should be incorporated into simulation during the design [26]-[28].

Using the new simulation tool, simulation of certain aspects of hospital reorganization was started at Ostrava University and of regional development as well. The simulation studies of hospitals reflect the patient-bed-fond managing with respect to use computer simulation for certain decisions demanded during the hospital operation [29]. The regional development simulation models take nto account that the region will organize consulting centers, where the inhabitants (or persons interested into immigration) will et information concerning the expected development of laber market, transport, housing etc.

XIII. Conslusion

The technique of nested simulating agents enabled to surpass the classification foreshadowed at the end of section V by the terms *models of order one* and *models of order two*. It was already the case of simulation of competing simulating systems [24] mentioned in the preceding section, where something like a ***model of order three*** was realized. We can speak of the ***depth*** of model levels. Beside it, another criterion concerns the number of (different) models – e.g. a computer can represented in a simulation model as an entity carrying more than one model. That is just a case of container yard simulation mentioned above, where the computer managing the yard handles two different simulation models, namely that of the fictive model, used in Φ for computing the shortest path, and that identified m, applied for testing the security of the computed path. We can speak on the ***size*** at different model levels. Nevertheless, the size immediately carries a further criterion into the possibilities of model nesting, namely the aspects whether there is a reflective simulation or not. For such a classification it is difficult to find words, as in connection with depth and size a complicated graph of the reflective relations can be thought. In [30] a certain attempt to classification was depicted.

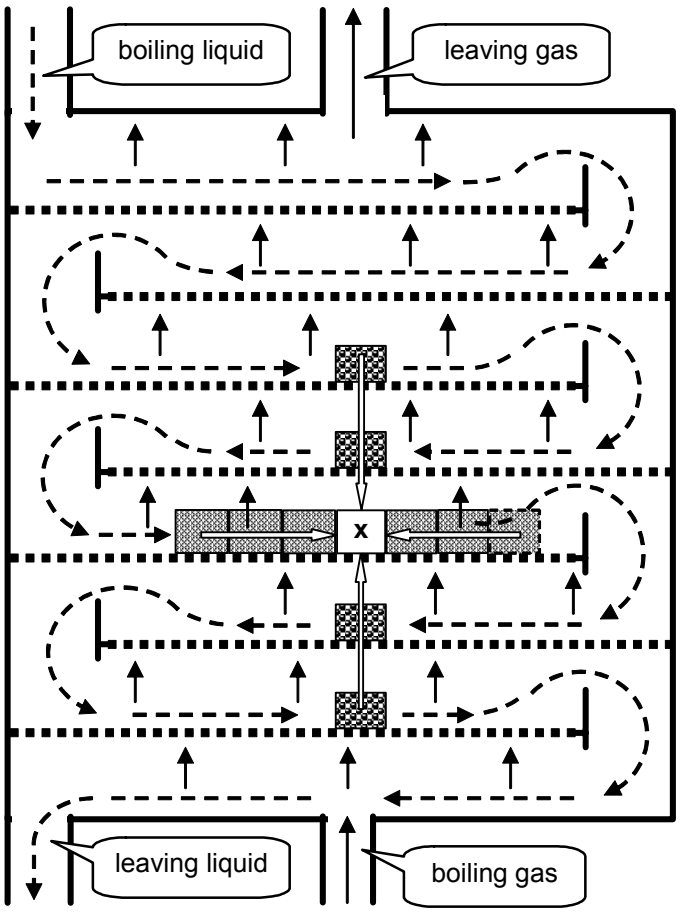

Fig. 6 Scheme of a rectification column simulation model – the environment of place x is occupied by agents that try to "bargain" the attributes of place x so that they correspond the partial differential equations describing the column behavior.

Nesting simulating agents may be a mental stimulus for further development in a larger domain, outside simulation. In general, "shaving" the common time axis from simulating agents is no problem and so one gets interesting cases, namely of nesting intelligent agents that do not need to be simulating. In Fig. 6 there is a schema of a simulated rectification column where certain agents are depicted as shadow squares; when they are just computing the values at place *x*: each of agents viewed some space or time development in directing to *x* (from left, from right and from the history), makes a certain approximation of possible true values in *x*, and then he agents perform a certain "discussion" to balance (and conciliate) the differentiating prognoses. No time scheduling exists during that discussion.

REFERENCES

[1] R. Rosen: *Anticipatory Systems*. New York: Pegamon Press, 1985

[2] G. Gordon: "A general purpose systems simulation program". *Proceeding 1961 EJCC*, New York: MacMillan, pp. 81-88

[3] T. J. Schriber: *An Introduction to Simulation using GPSS/H*, New York: Wiley, 1991

[4] O.-J. Dahl and K. Nygaard: *SIMULA, a Language for Programming and Description of Discrete Event Systems*, 5th ed. Oslo: Norsk Regnesentralen, 1967

[5] O.-J. Dahl: *Discrete Event Simulation Languages*. Oslo: Norsk Regnesentralen, 1966. Reprinted in [6]

[6] F. Genuys, Ed.: *Programming Languages*. London – New York: Academic Press, 1968

[7] 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

[8] O.-J. Dahl, B. Myhrhaug and K. Nygaard: *Common Base Language*. Oslo: Norsk Regnesentralen, 1968 (1st ed.). 1972 (2nd ed.), 1982 (3rd ed.), 1984 (4th ed.)

[9] *SIMULA Standard as Defined by the SIMULA Standards Group*. Oslo: Simula a.s., 1989

[10] O. L. Madsen, B. Møller-Pedersen and K. Nygaard: *Object-Oriented Programming in the Beta Programming Language*. Harlow – Reading – Menlo Park: Addison Wesley, 1993

[11] C. Herring: "ModSim: A new object-oriented simulation language". *SCS Multiconference on Object-Oriented Simulation*. San Diego: The Society for Computer Simulation, 1990

[12] Naur, P., Ed.: "Revised report on the algorithmic language ALGOL 60". *Communications of the ACM*, vol. 6, no.1, pp. 1-17, Jan. 1963.

[13] E. Kindler: "Transplantation – what causes it in MS-DOS SIMULA?" in *Object Oriented Modelling and Simulation of Environmental, Human and Technical Systems – Proceedings of the 24th Conference of the ASU, Salzau (Schleswig Holstein, Germany)*, B. Breckling and H. Islo Eds. Kiel: Ecology Center, 1998, pp. 155-164

[14] E. Kindler: "Reflective simulation in SIMULA," in *Applications of Distributed and Graphical Simulation – Proceedings of the 19th Conference of the ASU*, R. Kerr, Ed. Aberdeen: Kings College (University of Aberdeen), 1993, pp. D-2-1 – D-2-11. Reprinted in [15]

[15] E. Kindler: "Reflective simulation in SIMULA". *ASU Newsletter*, vol. 22, no. 1, pp. 1-14, Jan. 1994

[16] I. Krivy, E. Kindler and A. Tanguy: "Software for simulation of anticipatory production systems". *International Journal of Computing Anticipatory Systems*, vol. 11, pp. 320-335, 2002

[17] J. Mejtský and E. Kindler: "Diagrams for quasi-parallel sequencing – Part I". *SIMULA Newsletter*, vol. 8, no.3, pp. 46-49, Aug. 1980

[18] J. Mejtský and E. Kindler: "Diagrams for quasi-parallel sequencing – Part II". *SIMULA Newsletter*, vol. 9, no.1, pp. 17-19, Feb. 1981

[19] E. Kindler and M. Brejcha: "An application of main class nesting – Lee's algorithm". *SIMULA Newsletter*, vol. 13, no.3, pp. 24-26, Nov. 1990

[20] E. Kindler: "Classes for object-oriented simulation of container terminals," in *Managing and Controlling Growing Harbour Terminals*, E.Blümel, Ed. San Diego, Erlangen, Ghent, Budapest: The Society for Computer Simulation International, 1997, pp. 175-278

[21] 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

[22] E. Kindler: "Reflective simulation – first experiences," in *Simulation und Animation für Planung, Bildung und Präsentation '96*, P. Lorenz and F. Breitenecker Eds. Magdeburg – Wien: ASIM, 1996, pp. 39-50

[23] 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

[24] 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

[25] P.Bulava: "Transport system in Havirov," in *Proceedings of 28th ASU Conference, 2002*. Brno, Czech Republic:Technical University, pp. 57-62

[26] E. Kindler, P. Berruet and T. Coudert: "Conveyors with rollers and their reflective simulation," in *International Workshop of Modelling & Applied Simulation MAS 2003*, A. G. Bruzzone and R. Mosca, Eds. Genoa: McLeod Institute of Simulation Science, pp. 147-152

[27] E. Kindler, T. Coudert and P. Berruet: "Component-based simulation for a reconfiguration study of transitic systems", *SIMULATION*, vol. 80, no. 3, pp.153-163, March 2004

[28] P. Berruet, T. Coudert and E. Kindler: "Conveyors with rollers as anticipatory systems: their simulation models," in *Computing Anticipatory Systems CASYS 2003 – Sixth International Conference,* D. M. Dubois, Ed. Melville, New York: American Institute of Physics, 2004, pp. 582-592

[29] 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

[30] E. Kindler, I. Křivý and A. Tanguy: "Object-oriented sytem analysis of anticipatory systems in week sense". *International Journal of Computing Anticipatory Systems*, vol. 14, pp. 271-285, 2004