UNIVERSITY OF OSTRAVA

Institute for Research and applications of Fuzzy Modeling

Reflective Simulation - Simulation of Systems That Simulate

Evžen Kindler

Research report No. 85

2005

Submitted/to appear:
ESM - European Simulation and Modeling, Porto (Portugal), 2005

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.: +420-59-6160273 fax: +420-59-6120478
email: evzen.kindler@osu.cz

Reflective Simulation - Simulation of Systems That Simulate

Eugene Kindler

May, 2005

Ostrava University, Ostrava, Czech Republic
E-mail: kindler@ksi.mff.cuni.cz

KEYWORDS: Nested models, Reflective simulation, Object-oriented programming

ABSTRACT

The paper is dedicated to simulation of intelligent systems that "imagine" their possible future states by medium of simulation models handled by some of their elements. The reason and importance for simulation of such systems, the techniques for implementing their "nested" simulation models and the obstacles related to the implementation are described and some applications are presented.

1. Introduction

Intelligence is a concept, rooting in the human activities but then generalized, among other for phenomena of computer science; so one speaks on intelligent agents, intelligent control etc. The intelligence in that sense often replaces the human intelligence, models it and/or amplifies it. Under the term *intelligence*, one usually considers logical inference, adaptation, pattern recognition etc. Nevertheless, our own human experience reveals imagining as a very important component of the sort of thinking that is called intelligent.

Imagining can overpass the natural laws and logic. Nevertheless, let us consider the professional technical imagining in systems (that occur in simulation), i.e. let us limit our consideration to imagining controlled by the reason. Very often, such imagining concerns processes that will unwind during the time, often into the future. We have to concede the computer simulation as a model and amplifier of the mentioned type of the human imagining. Let *S* be an intelligent system of the mentioned type, for the behaviour of which the imagining is essential (i.e. if the imagining were not in *S* the behaviour of *S* would essentially change).

There are a lot of such systems. Beside (groups of) humans in a lot of their situations, man-made systems that apply more or less continually simulation for affecting their operation are of the mentioned sort. Among the simulationists it is spoken that "simulation is the worst method to get information on the studied system but often the only exact way for such an aim". Therefore when it is clear that such a system *S* is well conceived (designed) and when it uses simulation for the mentioned aim, the simulation cannot be eliminated – otherwise *S* would be badly conceived, using that "nasty method of simulation" without reason. In the next text, the simulation model applied in the described manner be called ***internal model***.

Such a system *S* may be simulated before it physically exists (for example during the phase of its design). Let its model for that purpose be called ***external model***. It should reflect the fact that among the components of *S* there is an information processing element *C* able to carry, to construct and to keep running the internal model and to have use of it for governing its own environment in *S*. If the external model did not reflect these properties of *C* the next dilemma would be present: either the external model would give false results on the future reality of *S*, or *S* would be badly designed, using simulation without reason (the suppression of the internal model made in the external model could be transformed into the reality of *S*).

The conclusion is that the external model *M* of *S* should contain the internal model *m* of *S*, nested inside. In general, an instance of the internal model may be step by step created many times and in any case it can differ from the preceding instances, reflecting the fact that the situation in *S*, varying in time, has to be reflected in the instance as its initial state. Using a model like *M*, we speak on ***nested simulation***, expressing the pure fact that a model is nested inside another one, or we speak on ***reflective simulation***, expressing the fact that the internal model starts with reflecting the state of the external one. Reflective simulation is a special case of nested simulation.

Although the reflective simulation is a suitable tool for anticipating the behaviour of man-made systems equipped by a simulating computer, requirements for it may come when a system is simulated, in which essential influence of imagining made by real humans (drivers, operators, etc.) is supposed. The internal models reflect such an imagining.

2. Obstacles With Nested Simulation

Computer simulation is applied when the simulated system is rather complex. A program for a model of such a complex system is a complicated software product and therefore already since the fifties of the 20th century simula-

tion programming tools (SPTs) have been designed; their benefit resides in that their users have to describe something very similar to the simulated system and not the algorithm governing the computer model: such descriptions are automatically converted into simulation models. One of the most useful tools of many SPTs is automatic scheduling of events coming from rather different sources, often almost independently. The scheduling is based on the properties of Newtonian time axis and therefore the SPTs automatically introduce one such axis for every simulation experiment.

Unfortunately, in nested simulation at least two time axes must be for disposal: when the internal model arises its time axis should start to exist and govern event scheduling during the whole existence of the model; when the internal model disappears and is later replaced by another internal model, a completely new time axis should arise; but simultaneously with this arising, manipulation and liquidation of various time axes of the internal models, exactly one stable time axis of the external model should exist.

With the exception of the 3O-languages (see further), no SPT allows introducing more coexisting time axes. It seems to be one of the main reasons that the nested simulation is not in a common praxis, contrary to the fact that it appears very useful.

The way to surmount this obstacle without loosing the merits of SPTs consists in using programming languages that are simultaneously object-oriented, process-oriented and block-oriented (let us speak on ***languages with three orientations***, shortly on ***3O-languages***). By means of the classes, subclasses and methods offered by the object orientation, one can define a set σ of methods for unerring manipulation with a time axis and therefore for scheduling of events in a given model; the process orientation enables the user to describe the "***life rules***" for the objects related to different classes; and the block orientation enables to introduce blocks, i.e. program components with local entities (classes, subprograms and variables), and – in brief – to set them among the steps of the life rules.

Object orientation enables the user to specialize σ, i.e. to "tailor" it to a genuine "problem-oriented" simulation language L, oriented to a suitable set Z of systems and often using professional expressions similar to those used by the non-computer-oriented experts communicating on Z). The block B into which σ or its specialization was introduced as its local "possession" represents a description of a simulation experiment; when the computing process enters B a model M of the system described in B comes into being and dynamically progresses; a simulation experiment with M is being performed whenever the program run is being inside the block. Suppose B is a subblock, i.e. a block inside another block A: then access to the values local in B (i.e. to the components of the state of M) is allowed from any place of A while B (and thus M) can offer its own instantaneous values to A. When the computing process leaves B the experiment ends and M disappears (but the values transferred to A and assigned as its local values remain as long as A exists).

The time axis T introduced by σ into B exists exactly when M exists. When the computing process again enters B a quite new simulation experiment with its new simulation model (including a new T) arises.

Iterating entering and leaving such a block B may represent a simulation study, i.e. a sequence of simulation experiments (Strauss et al., 1967). But the 3O-languages enable to nest a block b into the life rules of a class Ω of elements, introduced in B. And b can be equipped by σ, too, or by a specialization of σ; thus when the computing process is inside it, a simulation experiment with a model m runs, so that m models a system described in b. This simulation experiment has its own time axis t that is independent of T. While T and M remain when the computing process remains in B, t and m can disappear when the computing process leaves b, and they can newly start to exist as completely new entities when the computing process returns into b, although M and T are the same entities as before.

The described nesting of b inside B is a good way how to nest model m into model M. The class that we designated Ω in the preceding text can represent a class of computers or of beings able to imagine. That class, being local in B, represents elements that exist in the same "world" as the other elements of the simulated system S, and in M the instances of S are interpreted as existing in time represented by T, i.e. together with the other elements represented in M. Nevertheless, the instances of Ω have a phase Π of their lives, in which they are able to create models like m and to handle them like each of them would have its own time axis t that has no virtue to T. In fig. 1 the locality in blocks and other relations among the objects and blocks are depicted by means of so called Mejtsky's diagrams, which are suitable graphical aids to understand complex relations in the run of the programs written in the 3O-langages – see (Mejtsky and Kindler, 1980, 1981); C is an instance of Ω.

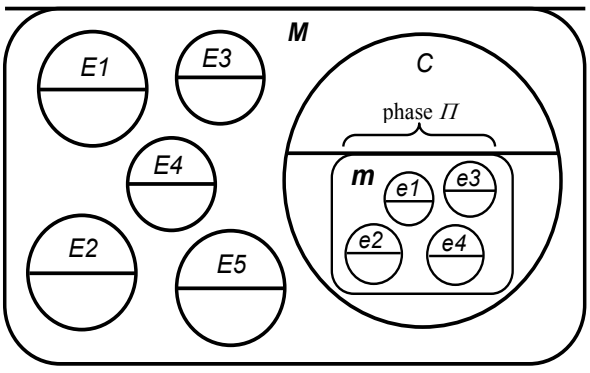

Fig. 1. Scheme of nested models: the circles *E1*,..., *E5* and *C* represent components of the external model *M*, their horizontal bisectors are metaphors for their "lives" (dynamics): move along such a bisector from the left to the right is a certain image of the progress of the "lives" during the time. The same holds for the upper edges of the "scenes" that represent blocks. The "life" of *C* (may be of a computer able to simulate) illustrates a state when a subblock has been entered, forming the internal model *m*, the components of which are *e1*, ... , *e4*.

3. Application of Non-Reflective Nested Simulation

Although it seems strange to apply nested simulation that is not reflective, i.e. to simulate systems containing elements that simulate something different from the same system or its part, we can meet that phenomenon in applications. It namely concerns the cases when the internal model realizes something that could be called ***pseudosimulation***. Let us explain this term.

A capable simulationist can view many phenomena as dynamic systems; among them, complex computing processes can exist and the simulationist can view them as interactions of parallel processes in a certain fictitious time; if his duty is to algorithmize such a computing process (and possibly to do that in a manner opened for unexpected modifications in future) and if he has access to a suitable SPT, he can change his duty to use the SPT and to formulate a description of the corresponding fictitious dynamic system F, i.e. to program a simulation model m of F: the run of the demanded algorithm would produce the same effects as an experiment with m. Naturally, the manipulation with m may be far from satisfying essential aspects of simulation (often time maps something that is far from the physical time) and therefore we speak rather on pseudosimulation.

Dahl (1966) was the first author who presented apposite examples of pseudosimulation – routines for computing the shortest path by using Lee's (or Dijskstra's) method, and for using Eratosthenes' sieve to get prime numbers. In simulation practice, we often meet systems containing elements that can be declared more or les intelligent and that need to perform rather complex computation. The routines for such a computation are to be nested in the "life rules" of such elements. Of course, computing the shortest paths is a transparent example of that and in case the routine is implemented according to the Dahl's conception, one meets nested simulation: the simulation model M of a "real" system contains the model m of fictitious system F of multiplying pulses visioned by Dahl.

This technique was applied in simulation of production halls served by induction-guided carriages (Kindler and Brejcha, 1990), of container yards (Blümel et al., 1997, chapter 4) and of regional bus service (Bulava, 2002). In case of the container yards with ground-moving transport tools, models of very similar fictitious systems were also used: when an empty place p for a container is determined and neighbouring places are already occupied, a danger exists that by placing the container to p a barrier of containers will be rounded off, which would forbid the ground moving vehicles to enter a certain part of the yard (Kindler, 2000). The first applications of nesting fictitious system simulation models inside models of real systems were collected and described by Kindler (1995).

Among them a rather pregnant example concerns simulation of rectifiers (distillation columns) – see e.g. (Kindler, 2002). Their behaviour is described by a complex set Σ of partial differential equations, the solution of which is supposed smooth at every plate. This smoothness facilitates the computing. The plates of the column were discretized to cells and when the vector of the solution values has to be computed for a cell λ a certain game of fictitious "mathematicians" is activated: they watch a near neighbourhood and a certain near history of λ, using splines they extrapolate the watched values to the present state of λ and offer the extrapolations as approximations of the solution, competing to give values, which might conform with Σ (see Fig. 2). The fictitious world of the mathematicians has no material relation to the chemistry of the column and is modelled by a model nested inside the model of the column.

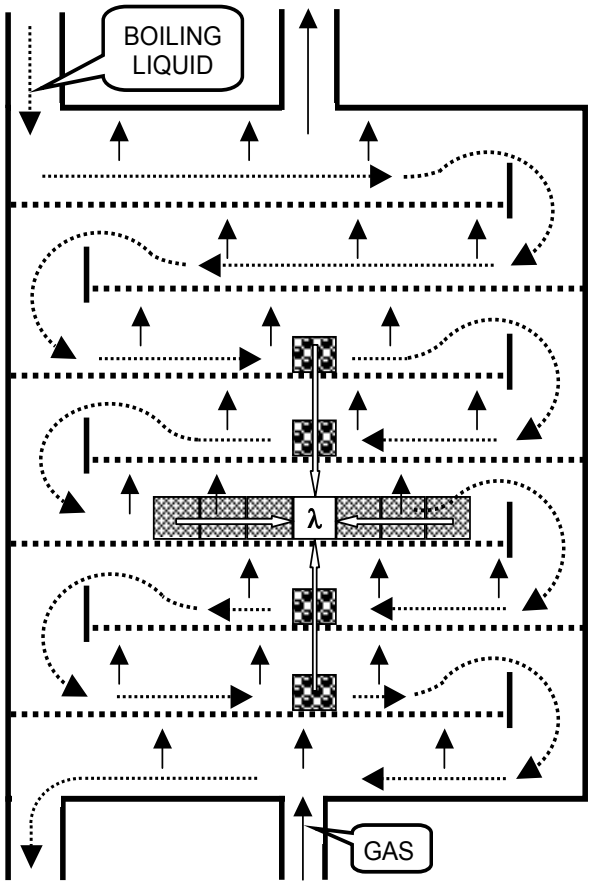

Fig. 2. Rectifier model scheme. The full arrows represent the gas flow; the dotted arrows represent the flow of the boiling liquid. The properties of cell λ are being computed: four triplets of marked cells around it represent four "mathematicians" working with the approximations estimated from the left, from the right, from above and from below. A similar approximation estimated from the past is not graphically represented.

There is also a case of inverse nesting, where a model M of fictitious session of mutually discussing experts $\{e_i\}$ controls a simulation study, namely a set $\{m_i\}$ of models that may concern real world; every expert e_i is modelled to have his own computer with a simulation model m_i, and – continuously during the discussing – he watches m_i, compares its behaviour with that of the other models, possibly modifies it during its run (or even refuses it, starting to experiment with a quite new model (Weinberger, 1987, 1988)). Model M has appeared excellent optimization software and – having use of the possibility to change the formulation of $\{m_i\}$ and the constraints to them – it was applied in a large spectrum of applications, starting from the industrial production (Wein-

berger and Mojka, 1983) and ending in biomedical domain (Faber and Weinberger, 1988).

4. Obstacles With Reflective Simulation

In chapter 2, it was told that a suitable specialisation of σ represents a genuine problem oriented simulation language *L*. In case the nested simulation is not reflective, the specialisation of σ for the description of *M* is far from that for the description of *m*. In case of reflective simulation, both the models concern similar systems (exactly: systems defined at the same "thing") and an opportunity exists to use something like *L* for description of the external model *M* and for the internal one *m*, too. It would be silly to suppose the use of rather different specialisations of σ for formulating *M* and *m*, i.e. to describe *M* in a language different from that in which *m* is described.

Satisfying this natural and rightful demand carries a hazard of a dangerous programming error, called transplantation. Its substance consists in erroneous mixing elements of different models. The simplest way to do this error is to assign an element belonging to a model *m* a name *N* determined for elements belonging to another model *M*.

Let us illustrate a possible progression of the consequences of such an error at an example of a patient-in-bed sector of a hospital. Suppose that the specialisation of σ respects that every patient *V* has two attributes, namely his *bed* and his *predecessor* (representing e.g. the last patient who entered the room before *V*), and that it is meaningful to apply

- *content* for every bed, representing the patient who is placed there (in case a bed is empty the value of content is *none*), and
- the (right hand) *neighbour* for any bed.

Suppose *L* is used for describing two different models *M* and *m* of the same patient-in-bed sector. In the following explication (but not in *L*!), let the elements of *M* be denoted by capital letters and the corresponding elements of *m* by lower case letters. A lot of other values can be computed at the basis of the attributes. For example, the (right hand) neighbour of a patient *Q* can be computed as the *content* of the *neighbour* of the *Q*'s bed.

In such sense, assume that model *M* should reflect the following state of the simulated system (see Fig, 3):

- *Q* is a patient returning from a certain treatment and should be placed at bed *B*;
- *B* is the neighbour of bed *A* where a patient *P* is placed, and *C* is the neighbour of bed *B*;
- patient *R* is placed at *C*;
- the patients entered the room in order *R*, *Q* and *P*.

Let the other model *m* be reflecting the same state and let the error consists so that bed *b* is assigned to *Q*. The imminent consequences are that patient *r* will figure as the *Q*'s neighbour and that *Q* will figure as the *p*'s neighbour. Not only two elements *Q* and *b* supposing to belong to different worlds, but already four elements *Q*, *b*, *p* and *r* are reshuffled into mutual relations. But the relation of *predecessor* engulfs other elements into the hodge-podge of the both models, e.g. both *P* (as the predecessor of *Q*) and *p* (as the predecessor of the predecessor of the *Q*'s neighbour). So after the mentioned "small" error the computing may proceed by regular steps but makes increasing chaos, handling together the elements of both the models.

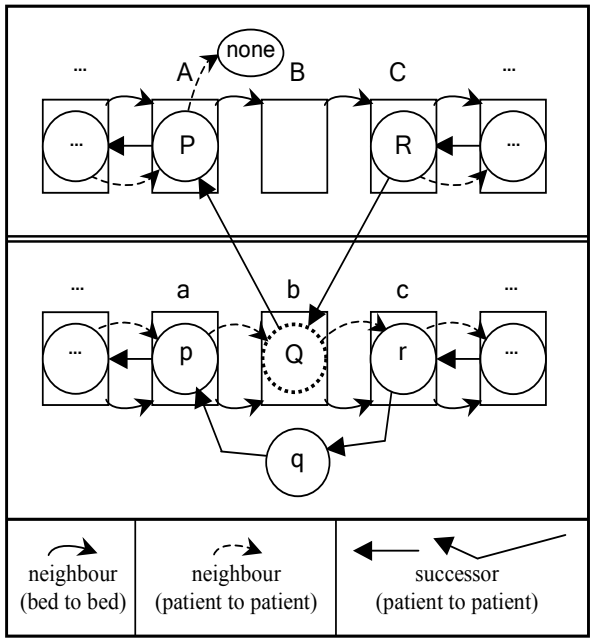

Fig. 3. Transplantation error and its near future consequences. Full lines represent values of attributes, dashed lines represent examples of values computed on the basis of the attributes.

According to what was just written, the chaos may seem being similar to what could follow an error in two-way list processing (a maze of predecessors and successors relations muddled among elements of many lists). Nevertheless, the chaos caused by transplantation is much more dangerous: the maze in the relations among the element of different models leads to chaos in the event evidence – those belonging to the time axes of both the models are chaotically mixed, which leads to a chaos of computing steps that should perform the elements of both the models and such a "wild run" continues until a fatal error (e.g. in applying a non-existing address), from which the reverse track leading to the first "small proper error of the author of the model" cannot be discovered.

There are only three 3O-languages that are implemented: SIMULA (Dahl et al., 1968) (Simula Standard, 1989), BETA (Madsen et al., 1993) and JAVA. Although nowadays JAVA is rather popular, it is not safe against transplantation, because its syntax is very free. Yet some occasions of transplantation are previewed but possibility of them is tested during the computing, which is one of the reasons that the models compiled from JAVA run slowly.

The syntax of SIMULA is rather limited; one of the reasons of that is to prevent transplantation. That is good, as all errors are detected already during the compilation and the

corresponding tests do not burden the model runs. BETA seems to be a certain compromise between SIMULA and JAVA, having rather free syntax and testing the critical events during the run of compiled programs. Unfortunately, the programs compiled from BETA are burden by the tests.

5. Use of SIMULA

Contrary to its "nominal" age of almost 40 years, SIMULA appears an excellent tool namely for its security and for quickness of the programs compiled in it. Its integral component is class called *SIMULATION*, which represents what we introduced in section 2 under symbol σ. Although this language appeared an excellent tool for simulation and for implementing problem-oriented SPTs without necessity to construct compilers for them, its security against transplantation seemed to be a property dear-bought at the cost of possibility to implement reflective simulation.

The security consists in that the models (i.e. the program components having use of *SIMULATION*) cannot receive names, i.e. cannot be identified; therefore it is not possible to use a technique sometimes called *qualification* (and in some object-oriented languages introduced under term *dot-notation*); if a value is identified e. g. as *x* both in models *M* and *m*, it is not possible to express "*x* of *M*" and "*x* of *m*" and thus to distinguish them. But such a distinguishing is necessary in forming the internal model, as it should start from a state constructed as a certain "copy" of the state of the external model, instantaneous at the moment of the demand to form and apply the internal model.

For example, if an element of the simulated system is represented under name *H* in both the models *M* and *m* and if we would like to transfer the numerical value of its attribute called e.g. *temperature* from *M* to the initial state of *m*, we would like to write something like (in SIMULA syntax)

$$m.H.temperature:=M.H.temperature \qquad (5.1)$$

but it is not possible. Another example that looked like it could not be formulated by means offered by *SIMULATION* is the following statement:

> "During the time interval $<T_1,T_2>$ the computer (a component of system *S* simulated in the external model) simulates in the internal model, what could happen is *S* during the time interval $<t_1,t_2>$."

The synthesis of the mentioned three orientations, which exists in the 3O-languages, is a revolutionary step in the development of the formal systems. The 3O-languages can be compared with formal theories able to generate and handle other formal theories and able to manage them to interact, admitting dynamic representation of their entities. Such theories were studied neither by logic nor by mathematics and therefore the 3O-languages, being without theoretical support of the common exact sciences, represent a branch plenty of surprises. Cognizing them is like cognizing powerful mathematical theories rooting in some axioms and definitions that – gradually in time – appear so fruitful that a lot of non-trivial consequences can be discovered.

So the SIMULA users thought during almost 25 years of its existence that the troubles mentioned above are essential. But in 1993 a method was discovered how to overcome the limitation (Kindler, 1993). It needed a certain sophisticated wit; discovering the wit was a certain analogy of discovering and proving a new and unexpected theorem in mathematics – once it is known it can be used for further development. The wit consists in furnishing the simulated computer *C* by one or more methods, the names of which are the same as those of the entities identified by the same names in both the models, and the results of which are those of the external model. The essence of the wit is chalked out in fig. 4 presented by means of name *X*. *B* is the block corresponding to model *M* and *b* is the block corresponding to model *m*.

Note that in practice it is not necessary to introduce such auxiliary methods for every name used in both models: after the mentioned way is constructed for one pair of elements,

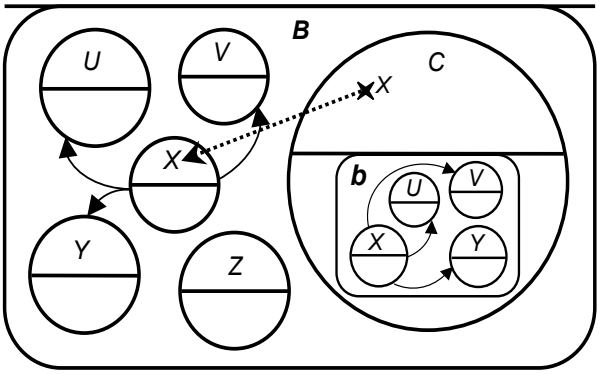

Fig. 4. A "naked" identifier *X* designates *X* of *m* whenever occurs in the description *B* of *m*, otherwise it designates *X* of *M*. It is true in *C* (but outside the description *b* of *m*), too. Therefore a method called *X* and introduced for *C* (so that *C* can perform it whenever gets a message like "*X* of *C*") can be programmed so that the result of it is *X* of *M*. The message can be applied anywhere inside *B*, and therefore inside the b, too, where it permits the access to *X* of *M*. So "*X*" and "*X* of *M*" can communicate inside *m* but – as "*X* of *C*" represents a result of a function and not a variable – no assigning for "*X* of *C*" is possible.

then many other entities are often accessible by means of pointers leading to them from those elements; see curved arrows in fig. 4, leading from the elements *X* – if e.g. *U* is formulated as *X.mother* (in SIMULA: *mother* of *X*) then inside *b* the age of *U* can be copied from the external model into the internal one by *X.mother.age:=C.X.mother.age*.

6. Applications

SIMULA (and the wit just mentioned) was applied in several situations, namely in the branch of the container yards, in that of circular conveyors and in that of simple queuing systems with intelligent control. Let us mention some details of those applications.

6.1 *Container Yards*

On the nesting simulation of container yards a mention was already made in section 2. The objective to form a universal simulation model of container yards opened to a spectrum as large as possible, i.e. to a spectrum of parameters concerning not only the quantitative aspects but also combinatorial and control ones. Simulation discovered a

deadlock danger in any container yard using two or more ground-moving vehicles for its internal transport, in case they do not intelligently anticipate their future. The substance of the danger consists in the fact that the computing of the (shortest) path of a vehicle is based on a certain (instantaneous) state of the yard but the application of the path takes some future time, during which the state often changes; a change can lock a place that is expected to be used at the path. A combination more locks is not excluded and sooner or later the simulation experiments fall asleep in states with totally frozen moving.

It is the drivers' anticipation (and that of the organizers), which protects the container yards against the deadlocks. A driver of a vehicle can see other vehicles and containers that could become barriers for his movement, and – according to watching their eventual activities (moving inclusive) – he imagines whether and how long they might be barriers. If he were able of a greater information processing (namely of remembering much more information on the past events and of much quicker deduction) his moving would be more effective. In simulation models, an ideal mode of the drivers' imagination can be modelled by simulation (and possibly used as a component of the automation of the yard control).

The simulation of container yards with such a generalized imagining was implemented during 1995-2000 under two projects of European Commission, controlled from the German Fraunhofer Institute for Factory Operation and Automation in Magdeburg. The external models simulate the container yards according to their material substance, while the determinination of the path of any vehicle is organized as a cycle of two phases:

- the shortest path *P* is computed as a sequence of places free of containers;
- simulation of the future is performed, using a nested model of the yard, where *P* is applied for the given vehicle *V*; the simulation experiment is concluded either (1) by the event when *V* accesses the target, or (2) by a conflict between *V* and a barrier. In case (1) *P* is assigned to *V* as its safe path. In case (2) a fictitious container is put at the place of the conflict and the cycle continues from its first phase.

The fictitious containers exist only in the nested models and are deleted as soon as a safe path is found. When a vehicle gets a safe path no future event can cause its modification. Theoretically, the described technique can finish by information that there is no safe path for a given vehicle and its given target, but in simulation of a bit realistic cases we did not met this case.

Note that the application uses two different kinds of internal models. The first phase is nested simulation that is not reflective (mentioned already in section 2) while the second one represents a "pure" reflective simulation (Kindler, 2000). This case illustrates that the relations between reflective simulation and non-reflective nested simulation may be more complex (Kindler, Krivy, Tanguy, 2004).

6.2 *Simulation of Circular Conveyors*

Another example of application is simulation of circular conveyors chalked out in fig. 5. The simulated systems are composed of a main cycle and of several working areas connected to the main cycle (in fig. 5, there are five such areas). The main cycle serves for transporting the objects ("parcels") to the working areas (from the entry place of the main cycle or from another working area) or to the place of exit. The parcels can rotate several times at the main cycle,

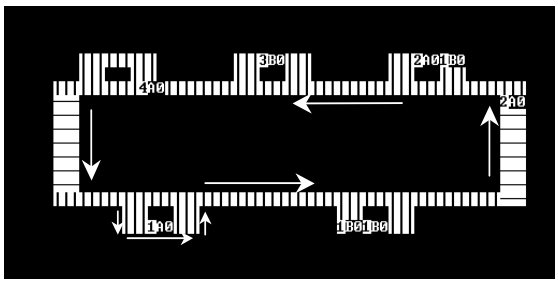

Fig. 5. A scheme of circular conveyor: the parcels are represented by boxes with 3 symbols (e.g. 1B0) representing the states of the parcels.

but to minimize that is one of the control objectives.

The important parts of the working areas are represented by small horizontal segments. The processing of the parcels is performed at its central place, one parcel can enter and wait for to be accepted for processing and one processed parcel can wait there to be allowed to leave the working area for the main cycle (in case of a danger of crash there). Suppose the conveyor is under design with a clear idea that when it works it will be controlled by a computer able to simulate; so it will be able to anticipate consequences of certain decisions and therefore to check their quality. In the simulation during the design phase, the mentioned tests for the quality were nested inside the used models as their internal ones. The following decision tests were included:

- A parcel comes to the conveyor in a situation when rather many parcels are placed there; should the parcel enter immediately to the main cycle or should it wait some time? Note that if it enters immediately it might uselessly complete a full cycle, obstructing other parcels to return from working areas.
- A fault comes, causing a working area inaccessible; should the fault be immediately repaired (at the cost of interrupting the complete conveyor functions) or would it be possible to continue its operating during a certain time, in order to finish a certain production task?
- A fault comes like in the preceding point; would it be possible to continue so that the function of the inaccessible working area(s) will be supplied by other ones (by i.e. by a "reconfiguration" of the system)? See (Kindler, Coudert and Berruet, 2004) or also (Berruet, Coudert and Kindler, 2004).

A very stimulating case, which belongs to the category of the reflective simulation of systems with automated operational transport was presented by Kindler, Krivy,

Lacomme and Tanguy (2004). Its complexity does not allow describing it here.

6.3 *Simulation of Simple Queuing System*

This example can be considered as a popular demo-instance of the reflective simulation models, as it concerns a system similar to that everyone encounters. It concerns a system *S* composed of transactions ("customers"), facilities ("tellers"), queues of waiting transactions and a dispatcher *D*. Time to time, *D* watches the queue lengths and may intend to lock a teller with rather short queue or open a locked teller (in case such a teller exists) in case the queues are rather long. But he has a computer, simulates the possible consequences of his decision and – according to the simulated data – he can change the decision (in attenuated or amplified sense). See e.g. (Kindler, 1996).

Using SIMULA for implementation of the model of *S* enabled interesting enrichments of it. So it was possible to model *D*'s computer *C* of a certain low rate and causing essential delay of the internal simulation: while *C* is simulating its environment changes. Moreover, once having described *D* it was possible introducing his "colleague" *d* (by a simple statement), and then to simulate *S* under different relations between *D* and *d*: the dispatchers can use computers of different rates and they can apply different criteria of decisions, different overlapping of their model runs and small or greater mutual communication (Kindler, 2002).

7. CONCLUSIONS

The wit mentioned in section 5 makes the reflective simulation possible but it would be suitable to try leading the practice of the reflective simulation to a simpler handling. An effort exists to arrange it, namely by forming a new version of the set σ (see section 2), that would allow giving names to the models. Having use of the fact that in order to get such a device σ should allow the models to be mapped as objects and not as blocks, the effort was fruitful (see fig. 6). The device is called *SIMULAT* and can be applied instead of standard SIMULA class *SIMULATION* mentioned above. Nowadays *SIMULAT* is tested concerning its efficiency and some other properties specific for SIMULA. Surprising is the fact, that the other properties of SIMULA make the use of *SIMULAT* quite safe concerning the transplantation, too.

To acquire a sufficiently large spectrum of experiences with the nested and reflective simulation, a lot of further models should be implemented in order to obtain a real image of that domain and to generalize the empirical evidences as truly as possible. The reflective simulation models of hospitals appear a fruitful step and it is expected that models of personal career can illustrate the reflective simulation from another viewpoint that those viewed from industry, logistics and queuing systems.

It is possible to nest an internal model μ into an internal model *m* nested in an external model *M*. In such a case there are three levels of simulation models inside a simulation study. The first steps were made in simulation of competing systems (Blümel, 1996) and the implemented model was like a pair <*S1*,*S2*> of those mentioned in par. 6.3. The essential contribution was in that the dispatcher *D1* of *S1* was modelled as simulating not only its environment, i.e. *S1* and *S2*, but – as a part of its interest about the future of *S2* – its simulating dispatcher *D2* (Blümel and Kindler, 1997).

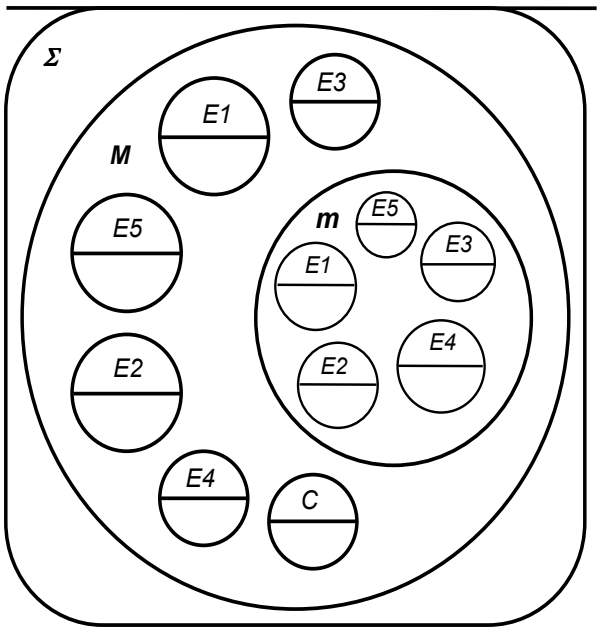

Fig. 6. Σ is a block corresponding to a simulation study. Inside it, the object *M* represents a simulation experiment and – may be said – the external model of a certain system *S*, containing the representations of the elements of *S* (including the computer *C* and the internal model *m*). The internal model reflects the elements of *S*, too. The statements like (5.1) can be simply used. For example, an attribute *x* of *E2* belonging to the external model can be read and assigned to the attribute *x* of *E2* belonging to the internal model by SIMULA statement *m.E2.x:=M.E2.x*.

A surprising way of the further development designated Novak (2000), embroidering the models mentioned in 6.1. He unified the fictitious "pulses" used in the Dahl's method applied in the non-reflective nested simulation phase, with the "elements modelling almost real transport tools" applied in the reflective simulation phase; the synthesis allowed getting better results than those mentioned in par. 6.1, i.e. computed by using the pair of the strictly separated nested simulation experiments; briefly said, the synthesis allowed to enrich the abstract pulses (figuring in the Dahl's method) by abilities viewed at the vehicles, among other by the ability to return; it enables a completely automated offering of new occasions for the vehicles to prevent a crash – while the method mentioned in par. 6.1 always ends in a deviation the Novak's technique offers what one often encounters in the real life: vehicle *V* approaching to a crash can bend aside and when the other actor of the possible crash departs *V* returns to its basic path; the deflecting and return may take less time than a deviation without return. The synthesis of abstract fictitious components with "real" ones, viewed and supposed in the combined elements, may become a fruitful and stimulating way for the future simulationists' thinking.

As the last aspect of the reflective simulation, it is to note a software system for automatic generating of reflective simulation models, the implementation of which was begun during 2000-2003 in a collaboration between Ostrava University Faculty of Sciences (Czech republic) and laboratory LIMOS of Blaise Pascal University in Clermont-Ferrand (France) under the bilateral convention on Barrande project system commonly superintended by Czech Ministry of Education, Youth and Sports and French Foreign Ministry. The given objective was an automatic translation of a (SIMULA) description δ of a conventional simulation model of a system S to a description Δ of a simulation model of S enriched by a class of "simulation professionals", i.e. of elements able to react to certain simple signals so that they detect all details of the state of S and generate a simulation model m according to the state they detected (Kindler, Krivy and Tanguy, 2003).

References:

Berruet, P., Coudert, T. and Kindler, E. (2004): Conveyors With Rollers as Anticipatory Systems: Their Simulation Models. In: D. M. Dubois (editor): *Computing Anticipatory Systems CASYS 2003 – Sixth International Conference, Liege, Belgium, 11-16 August 2003*. American Institute of Physics, Melville, New York, 2004 [AIP Conference Proceedings Volume 718], pp. 582-592

Blümel, P. (1996): *Mutual nesting of simulation models* [master thesis, in Czech]. Faculty of Mathematics and Physics, Charles University, Prague

Blümel, E. et al. (1997): *Managing and Controlling Growing Harbour Terminals*. The Society for Computer Simulation International, San Diego, Erlangen, Ghent, Budapest

Blümel, P. and Kindler, E (1997): Simulation of Antagonist Mutually Simulating Systems. In: O. Deussen, P. Lorenz (editors): *Simulation und Animation '97*. Society for Computer Simulation International, Erlangen, Ghent, Budapest, San Diego, pp. 56-65

Bulava, P. (2002): Transport System in Havirov. In: *Proceedings of 28th ASU Conference – The Simulation Languages*. FIT VUT, Brno, pp.57-62

Dahl, O.-J. (1966): *Discrete Event Simulation Languages*. Norsk Regnesentralen, Oslo. Reprinted in (Genuys, 1968)

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

Faber, J. and J. Weinberger, J. (1988): Thalamocortical Reverberation Circuit Simulation Using the Simula Language, *Acta Universitatis Carolinae Medica*, **34**, 149-248

Genuys, F (editor, 1968): *Programming Languages*. Academic Press, London – New York

Kindler, E. (1993): SIMULA and concurrent engineering. *ASU Newsletter*, **21**, 1993, No 3, pp. 1-16

Kindler, E. (1995): Simulation of Systems Containing Simulating Elements. In: M. Snorek, M. Sujansky and A. Verbraeck (eds.): *Modelling and Simulation 1995, Proceedings of the 1995 European Simulation Multiconference* [Prague, June 1-5, 1995]. Society for Computer Simulation International, San Diego, 1995, pp. 609-613

Kindler, E. (1996): Reflective Simulation (First Experiences). In: *Simulation und Animation für Planung, Bildung und Präsentation '96*. [ASIM Mitteilungen, Heft Nr. 54], ASIM, Magdeburg – Wien, 1996, pp. 39-50

Kindler, E. (2000): Nesting Simulation of a Container Terminal Operating With its own Simulation Model. *JORBEL (Belgian Journal of Operations Research, Statistics and Computer Sciences)*, **40**, No. 3-4, pp. 169- 181

Kindler, E. (2002): When Everybody Anticipates in a Different Way ... . In: Daniel M. Dubois (editor): *Computing Anticipatory Systems CASYS 2001 – Fifth International Conference, Liege, Belgium, 13-18 August 2001*. American Institute of Physics, Melville, New York, 2002 [AIP Conference Proceedings Volume 627], pp. 119-127

Kindler, E., Coudert, T. and Berruet, P. (2004): Component-Based Simulation for a Reconfiguration Study of Transitic Systems. *SIMULATION*, **80**, 2004, No. 3, pp.153-163

Kindler, E. and Brejcha, M. (1990): An application of main class nesting – Lee's algorithm. *SIMULA Newsletter*, **13**, No.3, (November), pp. 24-26

Kindler, E., Krivy, I., Lacomme, P. and Tanguy, A. (2004): Reflective Simulation Models for Optimization of FMS. In: *European Simulation Multiconference ESM '2004 [Paris, October 25-27, 2004]*. Eurosis, Ghent, pp. 94-98

Kindler, E., Krivy, I. and Tanguy, A. (2003): Automatisation de la construction de modèles pour la simulation reflective (in French). In: Y. Dallery, J.-C. Hennet, P. Lopez (Editeurs): *MOSIM'03 – Organisation et Conduit d'Activités dans l'Industrie et les Services [Actes de la 4ème Conférence Francophone de Modélisation et SIMulation 22-25, Toulouse, France]*, SCSI & CNRS LAAS, Toulouse, France, pp. 379-384

Kindler, E., Krivy, I. and Tanguy, A. (2004): Object-Oriented System Analysis of Anticipatory Systems in Week Sense. *International Journal of Computing Anticipatory Systems*, **14**, 2004, pp. 271-285

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

Mejtsky, J. and Kindler, E. (1980): Diagrams for Quasi-Parallel Sequencing – Part I. *SIMULA Newsletter*, **8**, No. 3, August 1980, pp. 46-49

Mejtsky, J. and Kindler, E. (1981):E.: Diagrams for Quasi-Parallel Sequencing – Part II. *SIMULA Newsletter*, **9**, No.1, February 1981, pp. 17-19

Novak, P. (2000): Reflective Simulation with Simula and Java. In: *Simulation und Visualisation 2000*. The Society for Computer Simulation International, European Publishing House, Ghent, pp. 183-196

Simula Standard (1989). SIMULA a.s., Oslo

Strauss, J. C. et al. (1967): The SCi continuous system simulation language. *Simulation*, **9**, No. 6, pp. 291-303

Weinberger, J. (1987): Extremization of Vector Criteria of Simulation Models by Means of Quasi-Parallel Handling, *Computers and Artificial Intelligence*, **3**, 71-79.

Weinberger, J. (1988): Evolutional Approach to Extremization of Vector Criteria of Simulation Models, *Acta Universitatis Carolinae Medica*, **34**, 249-258

Weinberger, J. and Mojka, A. (1983): Optimization of an Industrial Simulation Model by Means of Quasi-Parallel Handling, *Ekonomicko-matematický obzor (Economical and Mathematical Review*), **2**, pp. 179-185