Application of Computer Simulation in Information Systems

Cyril Klimeš
Eugene Kindler
Department of Informatics and Computers, Ostrava University
30. dubna 22, CZ–701 03 Ostrava,Czech Republic
Cyril.Klimes@osu.cz
ekindler@centrum.cz

Abstract
The paper concerns a project of implementing an intelligent information system. The intelligence should cover different aspects, namely reactions to some natural language directives and anticipatory self-organizing. At the start phase, programming tools that will be able to cover the simulation models of the design variants are made and tested. They are structured into four main levels, namely (1) that of the world with communication in fuzzy terms, (2) that concerning the world of repeated existence of systems (used for managing a simulation study), (3) that oriented to simulation of information systems (used for managing individual simulation experiments), and (4) that directed to the anticipatory abilities of the simulated systems. The programming tools are implemented in SIMULA.
Keywords: simulation, object-oriented programming, service enterprises, Simula, anticipatory systems

1 Introduction

There are many definitions of simulation but some of them are too limited (applicable only for a certain small domain) or too general (covering also some activities that more or less evidently exist behind the frontiers of simulation). The following definition, formulated according to (Dahl, 1964) appears to satisfy in the best manner what is understood under term simulation in the world professional manner: Simulation is a technique to replace a dynamic system by its simulation model and to perform experiments with this model in order to get information about the simulated system. Although it is more than 40 years old the development after its publishing demonstrated its accuracy.

In agreement with (Rosen, 1985), a person/team applying simulation for testing the operation and behavior of a designed system makes the person/team a computing anticipatory one; more perfectly and in agreement with (Dubois, 2000), a person/team applying simulation for testing the operation and behavior of a designed system is a computing anticipatory system in week sense; the used simulation model is really a formal model and serves essentially for computing of what could happen in the future.

When one designs an information system he anticipates that fact (that it is an information system that is to be realized and not a system of another type, e.g. a steel production, a hospital or an agriculture system). Even in this aspect, emphatic properties of computing anticipatory systems take place, which one can observe e.g. at object-

oriented knowledge representation, where the set of represented concepts (commonly known classes) is tested and used by means of a processor of an object-oriented language.

When one formulates a set of such a representations-classes, he has to anticipate the logic and mathematics applied in the domain in that he is engaged. Although the computers are manufactured to be applicable for the classical logic connected with the arithmetic of rounded real numbers, the other sorts of logic and computing are admitted. One has to anticipate what sort will be the nearest to the domain where he is engaged.

2 Description of the Studied Systems

The subject of the present paper is the design of an information system that would be used by Ostrava University for processing electronic documents and other information carriers. In the design, one must take a lot of future aspects into account, like the optimum number of hardware, the frequency of the requests coming to the system from its environment, the frequency of requests generated inside the system as a certain feedback inside the system, the structure of the University, the intelligent adapting of the hardware network to the instantaneous state of the requests, the laws reflected by the processing etc. It is possible to anticipate that all the mentioned aspects will change but for some of them one cannot anticipate the exact development of the changes. Nevertheless, one could anticipate at least some aspects of the mentioned developing properties, aspects that can be considered as "eternal" components of them.

Beside computer simulation, namely object-oriented programming is of use for that purpose. There are the following main qualifications of that programming paradigm in the mentioned objective:

(a) the object-oriented programming will be a good instrument for preparing simulation programming tools that allow to construct simulation models oriented to the hardware substructure of the designed information system so that one will be able to program the simulation models by the technique common in large scale simulation models, i.e. by describing simulated systems with certitude that the description will be automatically translated into simulation programs;

(b) the technique mentioned in (a) can adapt the simulation programming tools programmed by means of the object-oriented programming not only to new, unexpected factors that will appear in the future, but also some links of the hardware elements to the transactions handled by them as requests coming to the information system;

(c) already at the starting phase, the "eternal" factors of the views that we see to develop in future can be formulated and debugged.

Already in the present phase of the explanation, it is evident, that the simulation of the information system will meet three or more levels of world views (at least those corresponding to the mentioned in section 1) and so it is extremely useful to apply an object-oriented language that will be block-oriented, too. SIMULA (Dahl et al., 1968) (SIMULA Standard, 1989) appeared an excellent language; there is a free and efficient implementation for PC under Windows and under LINUX (transportable at the only one 3" disk and giving complex models that do not overpass 200 KByte).

3 First Level of Simulation

Because simulation is the first aspect observed as the instrument of anticipation, let the analysis of its tools (object-oriented representation of related knowledge) be described as the first stage. The anticipation of formal models of the knowledge proceeds in the steps described in the next subsections.

3.1 Nodes and Transactions

The information system will be considered as a network of permanent (or almost-permanent) nodes and tracks among them, along which ***transactions*** move. The time spent by the transaction at the tracks can be neglected. The transactions interact with the nodes so that they can wait in queues or take place of active "dialogue" with the nodes. Therefore any node has its certain capacity for the transactions present at the active dialogue, while – in respect to the present development of the information technology – the capacity for the queues can be abstracted as unlimited. Every transaction owns a certain item defining the node capacity necessary for its interaction with a node. Although one must admit that the item can change (namely that it can vary with respect to the properties of the node during the interaction), a rational stand-point is to consider it as a numerical value called ***size*** and to count with processes that can change it during the existence of a transaction. Such processes are not known at the starting phase of the analysis but the paradigm of the object-oriented programming permits to define them in the future.

Viewed by the sight of the present practice of simulation of queuing systems, the nodes can be qualified by means of common term ***storage*** and so it will be called in the next analysis. Nevertheless, it is suitable to let the formal term storage for richer and more realistic cases and let us call these starting very general notions as ***g_storage*** and – in the same manner – ***g_transaction*** (instead of transaction) and ***g_queue*** (for the concept of queue owned by a node), as well.

The first view to the instances of class *g_storage* can consider them as data structures composed of the components ***capacity***, ***free*** and ***queue***, while that to the instances of class *g_transaction* can consider them as having a numerical parameter ***size*** and being able to perform two procedures called (in line with common terminology of queuing system simulation) ***seize(S)*** and ***release(S)***: *seize(S)* should cover a rather complex activity, that the transaction demands storage *S* for an interaction so that if it is not immediately executable the transaction enters into the queue at *S*; when the transaction is turned to be executable the transaction finishes to perform *seize*. Performing *release(S)* denotes that the transaction that exercises it finishes its interaction with *S*, unblocks the part of the capacity of *S* and gives signal to the first transaction waiting in the queue belonging to *S* (if the queue is not empty), to try to start the interaction with *S*. Note the free part of the capacity of a storage is its component *free*; at the beginning, its value is assigned by that of *capacity*.

Class *g_queue* can be considered as ordinary class of ordered lists, very usual in simulation. The only special aspect is that we can think on its regime and introduce it

under a virtual procedure ***range(x)*** that ranges transaction *x* inside and can be prepared for FIFO-regime (first-in-first-out) by applying standard tools for list processing.

Class *g_transaction* can be specialized to many subclasses the instances of which can be present in the same systems and interact with the same storages. Therefore it is suitable to anticipate ***generators*** as sources of transactions of various classes, generating them with various frequency and assign them various properties, namely sizes. Therefore it is suitable to introduce class ***g_generator*** as covering the sources that generate transactions of class generally called ***new_transaction***, assigning them a certain ***size*** and holding during a time ***interval*** between generating two successive transactions. All the mentioned function *new_transaction*, *interval* and *size* are introduced as virtual, i.e. as meaningful in their context but without their proper content, anticipated as being formulated sometimes later, in subclasses of *g_generator*, namely according to the expected approach of transactions of various sorts into the studied information system.

3.2 Developing the First Running Models

The tools presented in 3.1 can be simply applied to construct the first running models. One can get names (e.g. *S1*, *S2*, ..., *S10*) to the nodes of the simulated system and to assign instances of class *g_storage* to them (e.g. by *S1:-new storage(3)*, to state that *S1* is the name of a node with capacity of 3 megabytes), the one can declare a class called e.g. *TR1* so that its instances are interested to interact with *S4*, then with *S7* and then with *S1*, each of the interaction taking *s*1.5* where *s* is the size of the transaction:

*g_transaction class TR1; begin seize(S4); hold(size*1.5); release(S4);*
*seize(S7); hold(size*1.5); release(S7); seize(S1); hold(size*1.5); release(S1) end;*

Note that SIMULA text could be shortened as follows:

*g_transaction class TR1; begin ref(g_storage)aux; real(x); x:=size*1.5;*
for aux:-S4, S7, S1 do begin seize(aux); hold(x); release(aux) end end;

And note that any algorithmic rules introduced for SIMULA could be applied in description of the life rules of a class of transactions, so that more or less complex and adaptive "lives" can be described by using cycles, branchings, assigning of reference values etc.

Accordingly also generators should be introduced, in the case mentioned above as

g_generator class generator_A(s1,s2,t); real s1,s2,f;
begin real procedure size; size:=uniform(S1,S2,U);
real procedure interval; interval:=negexp(t,U);
ref(g_transaction) procedure new_transaction; new_transaction:-new TR1;
end;

When one introduces *new generator_A(5,10,0.6)* he stands a generator that produces instances of class *TR1* with exponential distribution of frequency *1/0.6* so that the sizes of the instances are pseudorandom with uniform distribution between *5* and *10*. Naturally, more instances of such generators can be present (differing by their parameters) and other classes of generators can be declared using any algorithmic tools.

3.3 Animation and Data Collection

The best way to test the consistency and realism of the introduced classes (and to verify particular models) is to animate the simulation models and to observe their behavior at the computer display. Nevertheless, a suitable animation should run so that it is visually observable, i.e. rather slowly (moreover – its rate should be controllable by the modeler in real time on-line during the simulation model run), and a consequence of it is that when the animation is reduced to minimum the corresponding model runs much more slowly that the corresponding model that does not allow animation.

The models of realistic information systems will be large and with dense flow of events. Thus one should construct them, using well tested and debugged components (classes), i.e. with components that were tested also by animation but at much more simple models, the animation of which could give good and visual information. Therefore the best way is to prepare the basic classes (described in 3.1) in two versions:

The version supposed for animation is like that introduced in 3.1, but class *g_storage* is enriched by has two numerical attributes defining the position of every instance at the display, and procedures that visualize the state of the instance (and its queue); they should be called in any case when the length of a queue or the value of free is modified (namely in procedures *seize* and *release*).

The version supposed for real applications cannot have the mentioned animation tools but it is suitable to include into it – already up from the start works – tools for data collection, which enables evaluating the global behavior of simulation experiment, i.e. of the simulated variant. It is suitable to collect for every storage its exploitation, for every queue its average length and for every transaction its "dead time", i.e. the time spent in queues. The classes introduced in 3.1. are to be completed by certain auxiliary attributes that allow to use SIMULA standard tools for computing time integrals (for the storage exploitation and for queue length) and for the accumulation of the dead time (for the transactions).

4 Level of Simulation Study

Simulation study is a sequence of simulation experiments. The importance of such a notion is known for a long time: among other, repeating simulation experiments enables eliminating random deviations of individual experiments and computing mean values and their dispersions, the iteration of simulation experiments enables optimization.

Nevertheless, in case of simulation of information systems, repeating of simulation experiments leads to new components of its content, namely those concerning the simulation controlled by external data sets.

The reason is that in case of information systems one needs to test a great spectrum of possibilities differing in the structure of nodes and in the life rules of transactions, and that one anticipates that the spectrum can change in a long term development (e.g. so that a change of a political law implies a change of the information processing). In such a situation, to interpret every structural change as a change of the model would be far from efficient applications. The only way is to develop general models in that the

node configuration, the classes of transactions (their attributes and life rules) and the classes of generators would be essentially controlled by data. In other words, if such a model would be used a particular set of data files can convert its behavior to that of a particular model.

There are certain dangers related to such models. It is commonly known that the reading and transformation of information from an external file into computing memory takes time that is greater beyond compare that the time of computing; in other words, if the external data files should be read for every simulation experiment the simulation study would be protracted *k*-times (in comparing with that without contact with the external data files), where k would be much greater that one-digit decimal integer.

Moreover, if the life rules of the transactions were directed from the external data files the protraction of the simulation study (even of one isolated simulation experiment) would be still much greater.

Therefore the decision is clear: the information carried by the external data files has to be transformed only once at the very beginning of the simulation study so that corresponding information will be carried in form of data structures placed in the computing memory, i.e. at the same medium where the running simulation program and data generated and processed by the simulation experiments will occur during the operation of the given simulation study. Let the data structures be called ***control structures***.

While the classes oriented to simulation experiments are computer representations of concepts interpretable in the physical world, the classes of the control structures differ; if one would like to interpret them in something similar to the real world, he should think on something "eternal", namely in sense of classical philosophy, i.e. on something that is independent of time and can "enter" different worlds that may arise and disappear in arbitrary time relations. In other words, the concepts represented by the classes of control structure are to be anticipated as being applicable in a certain – may be large – spectrum of worlds, each of which could enter into consideration though only one pf them will be implemented in the physical world. Nevertheless, that anticipation must be tested by computing, when it is built into a computer model; note that simulation study is like a model of a lot of worlds, each of which having its own (Newtonian) time flow.

After a lengthy analysis, we have chosen the control structures according to our imagining and tested them and their relations at a large set of simulation studies. The control structures and their context are presented in Fig. 1, where

a rectangle with double line edges represents an external data file,

a rectangle with single line represents an "eternal" law (or – in the terminology of simulation experimenting – a structure that is global for the whole simulation study),

a circle represents an element of the real world (or – in the terminology of simulation experimenting – a component of particular simulation model),

a "scene" (rectangle with rounded cusps) represents a world-viewing (or – in the terminology of programming – a block),

a broad arrow represents a time consuming transformation from an external data file to an internal data structure,

a full single-line arrow represents establishing a component of simulation experiment (always at its start, i.e. once for a simulation experiment), and

a dotted arrow represents an almost continuous influence of a data structure to a component of a simulation experiment.

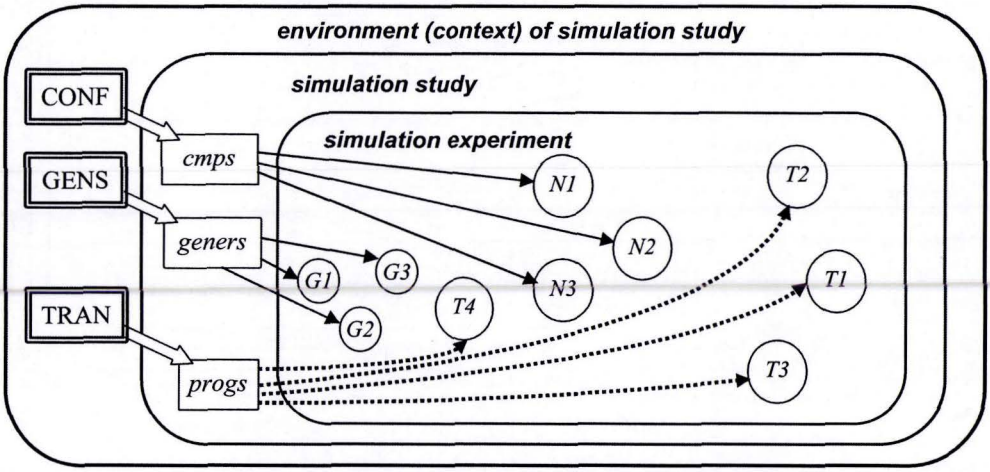

Figure 1: Control structures and their context

Namely, *CONF* is the data file with the configuration of the nodes, *GENS* is that with the properties of transaction generators and *TRAN* is that with the life rules of different classes of transactions, *cmps*, *geners* and *progs* are the "internal images" of the contents of the corresponding data files, *Gi* represent transaction generators, *Ni* nodes of the information system (computers) and *Ti* transactions. An illustration of application is in Fig. 2; it is a cut-out of a snapshot of display, where animation of an experiment, made

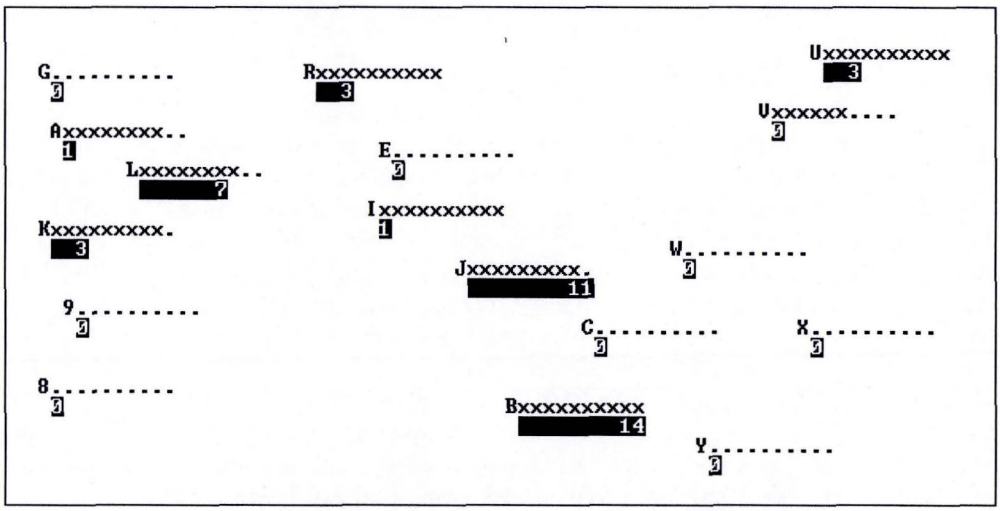

Figure 2: Part of a snapshot of animation

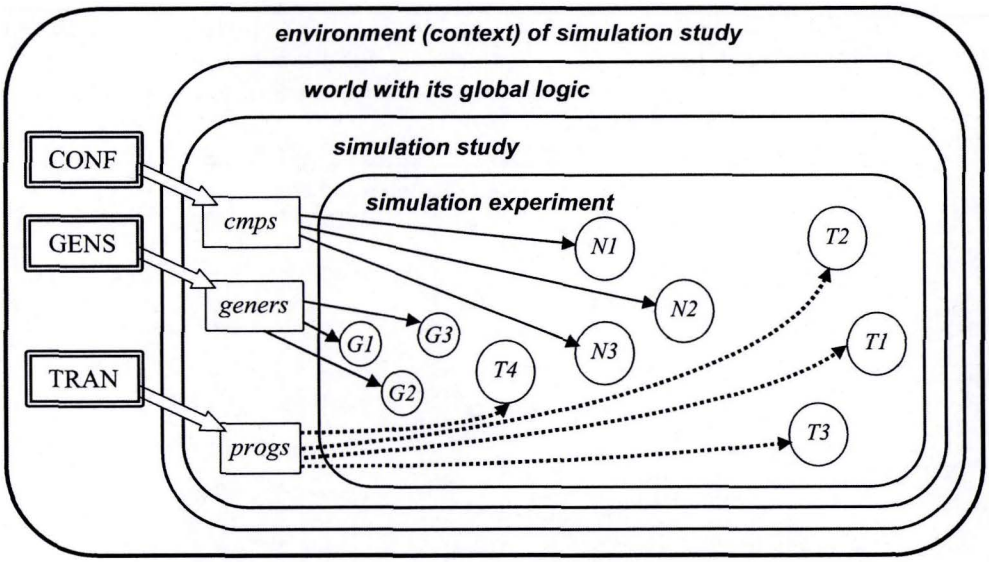

Figure 3: Level of "world with its global logic" included

in alphanumeric mode and concerning 36 nodes at all took place: the cut-of contains 17 of them, denoted as *A*, *B*, *C*, *E*, *G*, *I*, *J*, *K*, *L*, *R*, *U*, *V*, *W*, *X*, *Y*, *8* and *9*. The points represent the empty part of the capacity of a node, the letters *x* represent the part of the capacity just being occupied by transactions, the number of transactions waiting in the queue is underneath at the black background and in case their number is not too great it is represented by the length of the black background.

Note that SIMULA allows considering complex components of simulated systems (e.g. nodes composed with several "computers" with a common queue), but such details get over the frontiers and possibilities of the present paper.

5 Level of Global Logic

As was mentioned at the end of section 1, it is suitable to anticipate the arithmetic and logic applied in the designed information system(s) and to build the anticipation into the simulation software. One has to accept that the evaluation will generally be made according to more legal instructions, each of which being based on its particular logic (concerning namely the fine details). Therefore a profound analysis made in future is necessary, but already nowadays the optimal approach seems to introduce the highest level, i.e. a level in that all the above mentioned levels are nested and in that basic classes introducing something like fuzzy arithmetic and fuzzy logic are built. The virtuality of SIMULA procedures allows letting the classes open to adding new rules for the operations, equipping the level by the most probable sorts of arithmetic and logic, and by using several sorts of such arithmetic and logic during processing of the same transaction. The classes can be introduced according to the following scheme:

class g_fuzzy(a,b,c); real a,b,c; virtual:
procedure plus is ref(g_fuzzy)procedure plus(X); ref(g_fuzzy)X;;
procedure is_less_than is Boolean procedure is_less_than(X); ref(g_fuzzy)X;;
procedure minus ... ...

so that one will be able to write statements like

if mean_evaluation.plus(local_benefit).is_less_than(purview_f) then ...

The scheme of the world-viewings will be now enriched according to Fig. 3. Note that the tools mentioned in this section will be at disposal in any other level, too, therefore not only at the level of simulation experiment but at the level of simulation study, too.

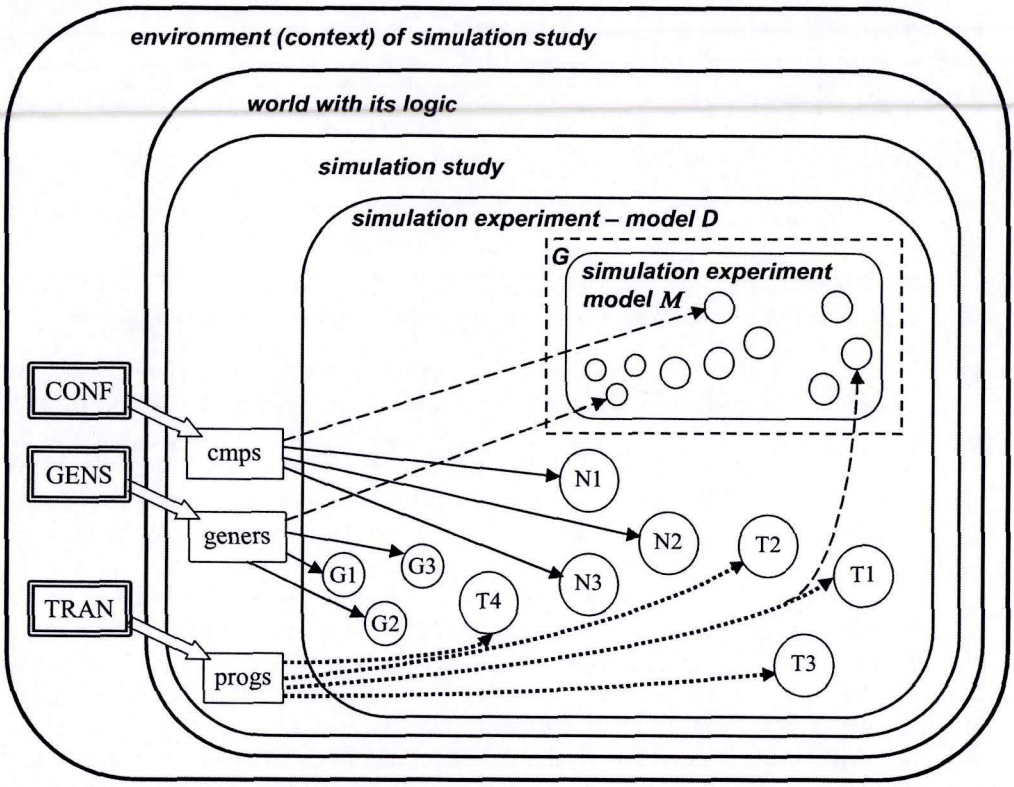

Figure 4: Level of the internal model is nested

6 Intelligent Reconfiguration

Any modern design of a system (not only that of an information one) should be disposed for automatic reconfiguration, i.e. fir an automatic change of its structure. Such an event can be taken into account for example when a fault attacks one or more permanent elements (in case of information systems: nodes) and a question arises whether immediately to repair the fault (even at the cost of disbarring a smaller or

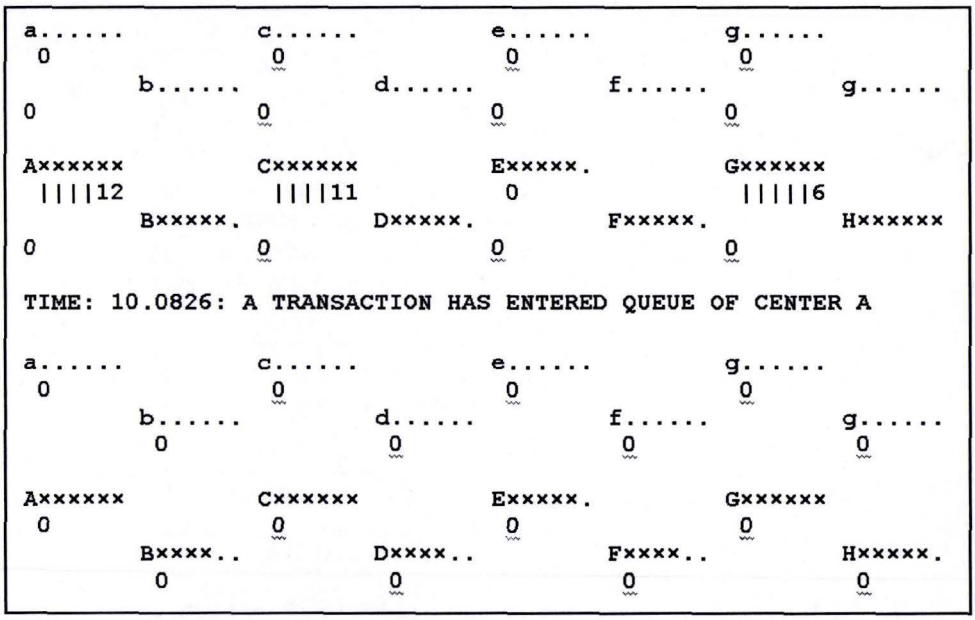

Figure 5: Part of a snapshot of an anticipatory computer system simulation

greater part of the system) or to continue during some time with a crank component. But the more intelligent system is the greater spectrum of stimuli for its reconfiguration (Berruet et al., 2004), (Kindler, et al., 2004). Testing whether to reconfigurate or not (and – in a positive case – how to do it) is a typical question concerning the future consequences of an instantaneous decision, and therefore the system that asks such a question to itself, is a typical anticipatory one and – similarly as many other anticipatory systems – it can apply simulation modeling to get answer.

The reconfiguration should improve the system behavior. In other words, we suppose reconfiguration when we expect that without it the system would behave in a worse manner, and – in order to satisfy the claims posed to it – that it should have other structure when it has to operate without possibility of reconfiguration (often the different structure consists in greater investments, but one can also think on different control algorithms etc,). Therefore, if during the phase of design of a system S an idea is being that later, when S exists and operates, it will reconfigurate, that fact should be considered as an integral ability of S and it should be built into the simulation model D which is used as an instrument to anticipate the S operating and development in the phase when it will physically exist.

For that purpose, the simulation model D should contain an element that is a certain image G of the simulating computer C existing among the other components of S. And that element G should be able to simulate like C, i.e. to detect the state of its smaller or greater neighborhood N, to reflect it as the initial state of a simulation model M constructed by the element, to activate M to run and – after a certain duration of the run

of M – it should process the information generated by M to get some support for decision whether (and possibly how) to reconfigurate.

D and M are in a certain sense similar, because both of them simulate similar systems S and N (it could be expected that frequently the unique difference between S and N will be the absence of G in N). Such a situation is called reflective simulation; although its realization (namely nesting of a model inside another model so that each of them have its own flow of simulated time and that the elements belonging to one of the models cannot be "transplanted" as components of the other one) seemed very difficult but at Ostrava University we have experience how to implement it, namely with an essential help of SIMULA block hierarchy and local classes (Kindler 2000), (Kindler et al., 2001). In Fig. 4 we can see the main scheme; note that the internal data structures are applied by both the models. In Fig. 5 one can see a cut-out of a snapshot of display, where animation (using alphanumeric mode) of a special application, where system of two vectors of nodes is simulated (see two rows of images at the bottom) and some decisions whether to use a node of the first or of the other vector is to be used. The anticipation of the consequences of particular decision is made by means of the nested model the images of its nodes are placed at the upper part of the figure.

7 Summary

Humans are anticipatory systems in weak sense (Dubois, 2000) and the formal models applied by them are systems of abstract notions at one side and systems reflecting the imagined "possible future". In science and technology, the last ones are controlled by rationality and thus they could be formulated by use of notions existing in the first sort of the mentioned formal models. When this mental activity of humans is transferred to computers, the systems of notions come into systems of classes formulated with use of object-oriented programming paradigm, while the imagined developments come into simulation models; the optimal way to implement such models is to program them by using the mentioned classes.

So it can be in case of anticipating during design of information systems. The team that designs an information system is an anticipation one, using computer models for simulation to know what could happen when a considered variant would exist and operate. Thus simulation not only helps to optimize the designed system but also gives stimuli that did not considered at the starting formulation of the variants.

Nevertheless, at the present time the information technology is so developed that the information systems on them owns should be considered as anticipatory systems, because they can use their own computing ability for to dynamically change and modify their properties (including their internal configuration). Among other, the information system can have use of their internal systems of notions and simulation models. That is to be included into the design of a new information system or of an essential modernizing/modification of an existing one.

So we meet nesting anticipatory systems, nesting classes and nesting simulation models: a team that designs an information system is an anticipatory one, basing its argumentation on another anticipatory system that is the information one.

The paper present certain initial steps in that way, including illustrations that such a programming of nesting simulation models is possible also in the domain of information systems design.

8 Conclusion

The starting works at the simulation modeling of intelligent information systems demonstrated the advantage of the structuring of the views into nested levels and introduced the authors into the uncommon and unusual analysis of the reality and synthesis of the models. A horizon for a question appears how the simulation software could be transformed to real time control of the implemented systems.

References

Berruet Pascal, Coudert Thierry and Kindler Eugene (2004) Conveyors With Rollers as Anticipatory Systems – Their Simulation Models. Computing Anticipatory Systems: CASYS 2003 – Sixth International Conference. Edited by Daniel M. Dubois, Published by The American Institute of Physics, AIP Conference Proceedings 718, pp. 582-592.

Dahl Ole-Johan (1964). Discrete Event Simulation Languages. Norwegian Computing Center, Oslo. Reprinted in (Genuys, 1968), pp. 349-394.

Dahl Ole-Johan, Myhrhaug Bjorn, and Nygaard Kristen (1968). Common Base Language (1st ed.). Norsk Regnesentralen, Oslo. 2nd edition 1972, 3rd edition 1982, 4th edition 1984.

Dubois Daniel M. (2000) Review of Incursive, Hyperincursive and Anticipatory Systems - Foundation of Anticipation in Electromagnetism. Computing Anticipatory Systems: CASYS'99 - Third International Conference. Edited by Daniel M. Dubois, Published by The American Institute of Physics, AIP Conference Proceedings 517, pp. 3-30.

Genuys Fernand (editor) (1968). Programming Languages. Academic Press.

Kindler Eugene (2000) Chance for SIMULA. ASU Newsletter, Vol. 26, no. 1, pp. 2-26.

Kindler Eugene, Coudert Thierry and Berruet Pascal (2004) Component-Based Simulation for a Reconfiguration Study of Transitic Systems, SIMULATION, Vol. 80, no. 3, pp.153-163.

Kindler Eugene, Krivy Ivan and Tanguy Alain (2001) Tentative de Simulation Réflective des Systèmes de Production et Logistiques. MOSIM'01, Actes de la Troisième Conférence Francophone de MOdélisation et SIMulation, 25, 26 et 27 avril 2001, Troyes, France: Conception, Analyse et Gestion des Systèmes Industriels. Edited by A. Dolgui et F. Vernadat, Published by Society for Computer Simulation International, Volume 1, pp. 427-434.

Rosen Robert (1985). Anticipatory Systems. Pergamon Press.

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