Application of Computer Simulation in Service Systems

Peter Vojtáš
Eugene Kindler
Department of Software Engineering, Charles University
Malostranské nám. 25, CZ–118 00 Praha 1, Czech Republic
Peter.Vojtas@mff.cuni.cz
ekindler@centrum.cz

Abstract
There is a problem to anticipate organization of services performed by an enterprise to its customer distributed in an array. At one part, the enterprise is interested to employ the minimum workers for that task, while at the other part the customers should be served as soon as possible after they send a message to the enterprise. Simulation of the variants viewed as materially possible, and then choosing the optimal one of them, is a good technique. A system that uses simulation is an anticipatory one and that anticipating the possible variants is also an anticipatory one, thus we meet nesting anticipatory systems. The anticipation of possible variants can be efficiently supported by applying object-oriented programming. That anticipation may pass over the design of one enterprise. The paper describes this technique and some illustrative examples.
Keywords: simulation, object-oriented programming, service enterprises, Simula, anticipatory systems

1 Introduction

When a team (or – in rare cases – a person) has to propose how to organize an enterprise it uses a model for evaluating the proposals. One of the important evaluations concerns income and according that evaluation it is to compare the variants and to determine the best (optimal or suboptimal) variant with the intention that this variant should be implemented in reality. The model could be intuitive (including a poor and simple imagining) or formal, having certain exact steps based on some rules forming a causal system. A team or a person that uses such a model during designing an enterprise is an anticipatory system according to popular definition by Rosen [1] or – in a more exact way – an anticipatory system in a week sense according to a more modern conception and definition by Dubois [2].

The formal models can be simple, suitable to be implemented "by paper and pencil" or by means of simple use of standard "office" computer software (e.g. spreadsheet processors). Although those models were often applied in operations research during the second half of the last century, they lost their importance, being replaced by simulation models running at computers. The main difference between the mentioned sorts of the models is as follows:

The causality among the steps performed by the simulation models reflects the causality among the events coming in the modeled (simulated) system, while the causality among the steps performed by the other exact models reflect another causality,

namely that obtained by human thinking using a projection of the view to the modeled system to a simplified human knowledge seizable by a human mind, and consequently by human computer-less analytical mathematical processing (this statement is not in a contradiction with the fact the results of such human-mind processing – like large sets of linear equations – could demand to be elaborated at the computing technique).

Because of the mentioned development, only computer simulation models will be taken in account further in the present paper.

There are many sorts of enterprise. This paper is oriented to centers of services (namely repairs) that have a set E of employees who left a center C for visiting a set S of customers, in order to satisfy their demands. S can be static (constant) or dynamic (customers may disappear and new customers can come), certain aspects of the moves of the employees can have a constant rhythm (called everyday one – e.g. their everyday gathering in C at the beginning of the shift) but the distribution of the demands of the clients (elements of S) surpasses any rhythm and causal relations, though some quasi-periodic repeating of events that could be observable and statistically evaluated can lead to certain rules for the moves of the employees. Let the mentioned systems be called ***systems with itinerating employees***, shortly ***IE-systems***. The task of the proposal to organize a team is often related to a steady state of the employment rate and so the set E of the employees is viewed as constant. Nevertheless, IE-systems can have more centers like C.

2 Everyday Problems in the Enterprise Systems

When an enterprise is designed it is to take its everyday problems and decisions implied by them into account. That was not made even when simulation was applied, contrary to the fact that a more or less bad decision can more or less deteriorate the enterprise operation; in other words, when a regime with worse decisions will be interpreted in the formal models during the design the results of the optimization could demand more employees, therefore more wages and therefore less income of the enterprise.

Evidently the service IE-systems offer a large spectrum of decision quality, especially because the problems of shortest paths combined with rather short of everyday rhythm period. It is known that such enterprises makes more or less exact argumentation in order to derive everyday scheduling of activities. One can observe that the enterprise (or a dispatcher of it) is an anticipatory system (in a week sense) that uses a model of the starting (or of the following) day activities in order to find some suitable way of them.

In case of the mentioned model, the situation is similar to that noted in the preceding section: from intuitive models based on imagining, analogies and simple budget, over application of scheduling computer programs until "short term simulation" In case of itinerant employees the situation is still more complicated; it is often necessary to determine the shortest path and to adjust it to the time demands and possibilities. Note that the computing of the shortest path can be based at special simulation models (see e.g. the last pages of [3]), while the adjusting to the real time possibilities could be a hard

problem (at our University, a problem was met, concerning scheduling tens of trucks that collect veterinary rests – namely from slaughterhouses placed in mutually distant places – and transport them into factories: it appeared to be a really hard problem [5]).

3 Nesting Models

Therefore, when one has to design a suitable IE-system and wants to use the best formal model for it, namely a simulation model, he tends to meeting *nesting simulation* (simulation of systems that contain simulating elements [6]). A simulation model M used in the design phase should contain an element D (e.g. an image of the dispatcher or his computer) carrying another simulation model m that corresponds to that applied in "everyday". As it was mentioned in the preceding section, D could have use of some other simulation models, e.g. for computing the shortest path. Such a model exists in the same context and environment as m; let it be called μ. A scheme of the nesting is in Fig. 1, where *f1*, ..., *f5* illustrate some components of M (e.g. images of objects that exist in the simulated system in the same manner as the dispatcher) and Σ represents a *simulation study* (iterating simulation experiments in order to get some special result like optimum configuration, average values, extreme values etc.). Note that the present paper neglects considering model μ.

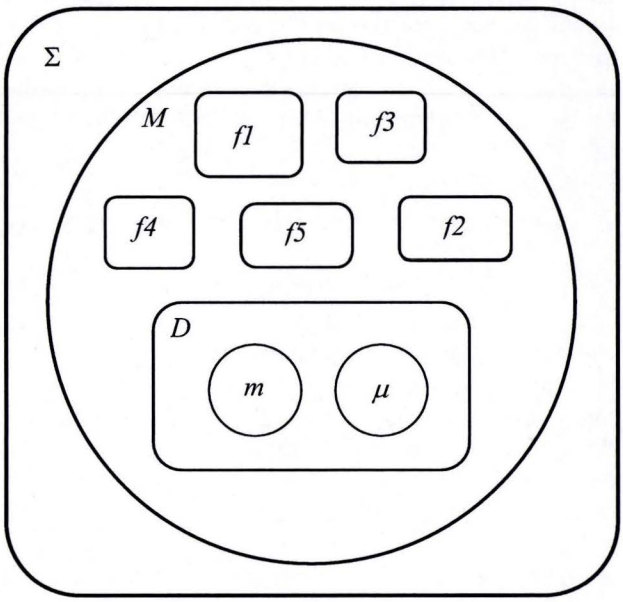

Figure 1: Scheme of nesting

In order to begin the works at the implementation, we used programming language SIMULA [7], [8], as it is extremely suitable for nesting simulation models, because of its object-orientation, process-orientation, block-orientation [9] and safe separation of the model, program and knowledge description from that concerning the simulating computer [10].

4 Anticipation of Future Simulation Programs

A person (or a team) that produces the simulation program π can be considered as a system P, which interacts with its environment E that formulates the demands

concerning π, namely the properties of the designed system interpreted in the simulation model. P may behave in a more or less intelligent manner. In case of extremely low intelligence, P, working on π, directly follows the instructions coming by E. The greater intelligence exists during the works on π, the greater anticipating of possible future modifications formulated by E, is taken in account. The anticipation can even lead behind the frontiers E and takes into account possibilities that π would be applied in another occasion, i.e. in case its environment that formulates the demands will be quite different from E (e.g. in case P anticipates to utilize π in designing another enterprise, that could be not yet known to the given time).

Object-oriented programming offers instruments such an anticipation, i.e. offers instruments to turn P into a computing anticipatory system: P could formulate concepts common for all anticipated variants of dialogues with E and represent them at computing technique so that in case of any new variant of E comes it will be simply language transformed into the corresponding π (said in other words: P should anticipate all forms of the – natural but professional – language L and represents its vocabulary and grammar structures on computer so that any new demand expressed in L could be simply written on the computer input and then automatically taken as the simulation program π corresponding to the new demand formulated in L). By means of corresponding language processor (usually a compiler or an interpreting program), which processes more or less immediately the formulations of the concepts-classes, a good object-oriented programming language (like SIMULA or SmallTalk) discovers any contradiction and incompleteness during representing L on the computing technique.

Thus we come to the third level of anticipatory systems. A start base of the language L corresponding to the enterprises treated in the present paper is outlined in the next two sections.

5 Level of Simulation Study

As it was already mentioned, simulation study is an iteration of simulation experiments. While a simulation experiment can be viewed as a computer model of a part of a certain (real or virtual) world, in which Newtonian time flows, simulation study can be viewed as a model of some entity that is eternal (in sense used by ancient philosophers), i.e. in which no time from exists but various "worlds" with their own time flows can arise and disappear. An experienced simulationist thinks about the possibilities whether the hierarchy "simulation study – simulation experiments" can be applied for making the human and/or computer work more effective. In applying object-oriented programming, the first idea concerns the "eternal" concepts of the future simulation studies, i.e. the concepts that are independent of alternating simulation experiments.

In case of subjects to that this paper is oriented, the "eternal" concepts are those concerning drawing at the computer display and generators of pseudorandom numbers (said in "user friendly" terms: the display does not disappear with the end of simulation experiment and ignorance of causes (which is in general the base viewing something as

randomness) is also an aspect that does not disappear or change with the end of a simulation experiment).

The representations of simulated elements at display are organized in three classes. The basic class *IM* (abbreviation of *image*) represents general image, that has its place given by coordinates *X* and *Y*, its form *FORM*, its color *COL* and Boolean function *equals* that enables in a simple way to express a test whether two images are at the same places (e.g. in test *if A.equals(B)*). In using SIMULA class *TERMINAL*, *FORM* is of type *character* and *COL* is of type integer, but the same class *TERMINAL* offers mnemonic names like *white*, *red*, *blue*, *brigt white*, *light red* etc. in place of integers by which the are the colors enumerated.

Class *IM* is specialized to four classes, called *IM_ST*, *IM_MV*, *IM_AU* and *IM_FT*. *IM_ST* represents concept of static image and has attributes *IM_LEFT*, *IM_RIGHT*, which point to instances of class *IM_AU*. This class represents the concept of auxiliary images; note that it appeared to have two auxiliary images for any static image at disposal, one of which at its left side and the other at its right side. *IM_FT* is a fictitious image (applied in special sophisticated handling). *IM_AU* and *IM_FT* have no added attributes and procedures in relation to *IM*. *IM_MV* represents concept of moving images and has procedure for moving to a place with given coordinates.

Class *G_RANDOMS1* is the basic class of generators of random numbers; it has one parameter *MEAN* and virtual function *VAL* that is expected to give the next pseudorandom value (for example, in case *G* is a generator, then *G. VAL* gives the next value and behaves like pressing button *VAL* at the vendor automaton *G*). Class *GEN_EXPON* is a subclass of *G_RANDOM* the *VAL* of which gives pseudorandom numbers with exponential distribution. *G_RANDOMS1* is also specialized to class *CONSTANT*, the *VAL* of which gives constantly the result *MEAN*.

Another subclass of *G_RANDOMS1* is *G_RANDOMS2* with added the second parameter called *SIGMA*. *RANDOMS2* is specialized to *GEN_NORMAL* the *VAL* of which gives pseudorandom numbers with normal distribution with mean *MEAN* and with standard deviation *SIGMA*. Similarly, *G_RANDOMS2* is specialized to classes *GEN_UNIF* and *GEN_INT* the *VAL* gives pseudorandom numbers with uniform distribution on interval <*MEAN- SIGMA, MEAN+ SIGMA* > – in case of *GEN_UNIF* they are real numbers, while in case *GEN_INT* they are integers.

For a certain comfort of the future users, a procedure was added that automatically assigns one character names to the just generated instances of *IM_ST*. Also some procedures enabling prompt description of initial dialogue defining the system constants have been added to this level.

6 Level of Simulation Experiment

Description of a simulation experiment is equivalent to a description of the simulated system, completed by simulated data collection and conditions for finishing the experiment. In our case, the description of the simulated system is expected to be programmed by means of classes formulated in object-oriented programming paradigm.

Therefore such classes should follow sorts of elements existing in the described, simulated system.

In our case, there are two main classes of that sort: *client* and *server*, where client represent a center demanding the enterprise to send a server to it in order to satisfy the demanded task (in general – repairing of something existing at the client).

The "life rules" (algorithm of activity) of class *server* is formulated as a cycle the steps are composed of two phases: the *passive* phase during that the server does not interact with the clients but accepts and collects their demands, and the *active* phase during that the server serves one or more clients, moving from one to another. For collecting the clients' demands, every server has a list called *demands* where the demands sent by clients are stored in a form of cards, i.e. instances of an auxiliary class *card* formulated so that every card carries a reference to a client that sent the corresponding message of demand.

Other attributes of *server* are those serving for accumulating the active and passive times, and *position*, which is a pointer to an *IM_MV* that represents the server's image position at the computer display and informs on its place (see the preceding section). Many procedures of *server* are specified as virtual, being prepared for a large spectrum of decision steps like ranging a *card* into *demands*, switching the *passive* phase to the *active* one, duration of the serving a client etc.

The "life rules" (algorithm of activity) of class *client* is also a cycle of steps composed of *active*, *passive* and *detached* phases. Explained by means of a metaphor, *client* makes some work during the *active* phase, for which he needs a certain "hardware"; finishing the work, he reposes during some time during the *passive* phase and then he returns to the active phase; at this moment, he could discover a fault at his hardware and then he sends a message to a server, demanding him to come and repair the fault; waiting to the end of repair, client is in *detached* phase.

Among the attributes of *client* are those serving for accumulating the detached time, and *position*, which is a pointer to an *IM_ST* that represents the client's image position at the computer display and informs on its place (see the preceding section). Another attribute represents the color that is transferred to *position*, in order to visualize the client's phase. Virtual procedures are specified for class *client*, serving to decide on the rise of a fault, for the duration of active and passive phases etc.

7 Examples

7.1 Distributed System According File Information

Certain commercial problems of designing the enterprise internal control concern the rules for the servers. So the first example was oriented to a system with one server and with a configuration of clients, the positions of which are read from a file. The server's passive phase is defined according to the number of demands: the phase ends when the server has got a certain number N of demands – then he leaves his passive phases and starts to the clients. He serves them in the same order in that he gets the demands. The duration of the active and passive phases of the clients and that of the repairs are pseu-

dorandom values, similarly as the probability of faults. The parameters of the corresponding distributions, the rate of the server's move and the value of N are given in the initial dialogue with the operator, and the duration of a simulation experiment and of the number of the experiments in a study as well. No special transport network is supposed.

In Fig. 2 there is a monochrome transformation of a snapshot taken from the display with a pseudo graphic animation. The letters and digits represent clients, crosses approximate the performed way of the server (usually only one cross is present and moves, modeling the server's moves.

7.2 Dining Philosophers Around Full Table

The next example connects to the popular system of chinese philosophers sitting around a round table and intending to eat. For starting to eat, each of them needs to seize chopsticks into every hand; one chopstick is placed in the space between two neighbouring philsophers so that any of them can seize it. When philosopher ends

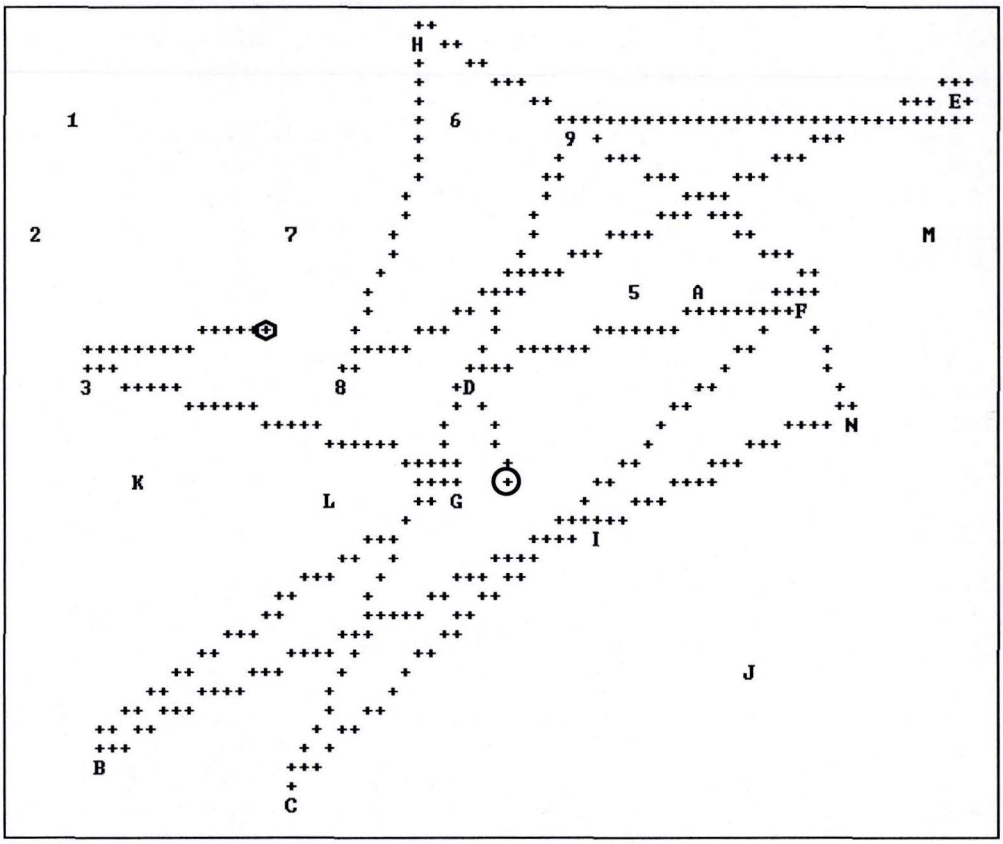

Figure 2: Snapshot of animation with pseudographically saved trace of a worker, starting from the place marked by the small circle, then visiting places denoted by D, C, F, H, 8, E, 9, D, A, F, N, B, G and 3 and reaching the place at the small hexagon.

eating he returns the chopsticks at their original places but with a certain probability he can drop a chopstick down. When a philosopher wishes eating and misses chopstics, the

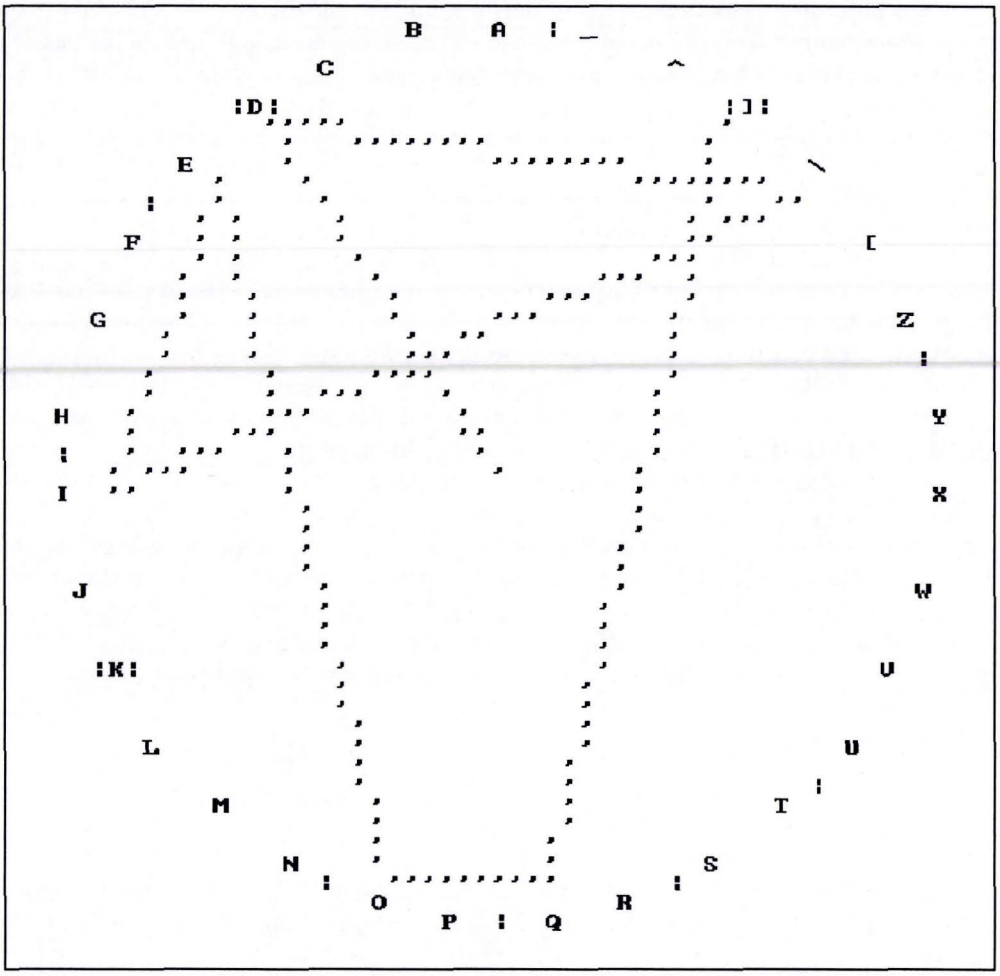

Figure 3: Snapshot of animation with pseudographically saved trace of the waiter, starting from the center of the circle, then visiting philosophers denoted by D, \, I, E, O, Q and]. Note that the waiter had sometimes to wait for a new task.

reason of it may be either that his neighbour is eating or that the chopstick has been dropped down. Assume no philosopher consider difference between those two variants. When a philosopher p has not a chopstick for disposal, he immediately or sometimes later demands the "waiter" W to come and give him the chopstick. W realize it only when the demanded chopstick has been dropped – then he lefts it and gives to p. When

the demanded chopstick is being handled by the neighbour, the *W* cannot help and considers the demand as a needless derangment by *p*.

The input parameters are given by the operator during the start of the simulation study and the number of philosophers as well. It is supposed that *W* spends his passive phase at a place rather distant from the table, and when it decides to serve the philosopher he makes it walking around the table and during it trying to serve the demanding philosophers. He passes without delay the philosophers who expressed no demand.

7.3 Dining Philosophers at Isolated Tables

A similar model of dining philosophers was also implemented, differing from that of 7.2 so that the philosophers are sitting at isolated small tables and, similarly, every chopstick is placed it another isolated table so that the chopstick is accessible by two neighboring philosophers. That configuration enables the waiter freely to move inside the circular configuration of the sitting philosophers and directly to move from one philosopher to another without touching the other philosophers.

In Fig. 3 there is a monochrome transformation of a snapshot taken from the display with a pseudo graphic animation. Apostrophes approximate the performed way of the waiter (usually only one apostrophe is present and moves, modeling the waiter's actual place). The chopstics are represented by small vertical incises |. When a chopstick is not occupied and lies at the table its representation is placed at the circumference of the circle. When the chopstick is being in a hand of a philosopher its representation is placed at the same line of the display as the symbol of the philosopher that takes it.

7.4 Experience With PC

Both the models mentioned in 7.2 and 7.3 were also modified so that e.g. the waiter in 7.2 does not count the messages coming to him, but instead he waits during certain time followed the first message and then goes to serve.

The simulation models realized in the described manner by using the implementation of language SIMULA for IBM PC compatible computers [8] appeared to work suitably quickly.

First consider the experiment with the model of dining philosophers.

Let *M* be the number of messages that the waiter accepts before he decides to serve the philosophers.

As an example, we present information on a simulation study that should test the optimal value of $M = 0, \ldots, 5$ in case of system with parameters mentioned in Table 1 (*t.u.* means a time unit common for any time data of the model, in practice one can understand it as e.g. 10 second), and for evaluating by one financial unit both the time unit spent by any philosopher by waiting and the time unit spent by the waiter during its work.

The distributions of duration of the time consuming phases are exponential.

Table 1: Parameters of experimenting

number of philosophers	31	mean time of eating	30 t.u.
mean time of resting	20 t.u.	probability of dropping chopstick	0.9
waiter's move to the table	5 t.u.	lifting a chopstick	1 t.u.
waiter's move from one philosopher to his neighbor	1 t.u.	duration of a simulation experiment	100000 t.u.

10 simulation experiments were performed for computing average values for any case of *M*, i.e. 60 simulation experiments of length 100000 simulated t.u. had to be performed (relating to the minimum prize of the sum of waiter's proper work and the philosophers' waiting times, the optimum appeared as *M=4*). At a cheap IBM PC computer with Celeron 466 MHz processor), the whole study needed 202 seconds, therefore only 3 minutes and 22 seconds.

We have also experimented with the model 7.1. For these experiments input parameters can look like in Fig. 4,

```
configuration from file:basic_cnf.asc
seed:123456789
average working time:30
average resting time:20
probability of spoiling in the machine:0.1
step-by-step? Put 'y' or 'Y'
distance between events, RT-seconds:0.2
distance between alarms:5
serviceman's time for one step:1
serviceman's time for rising a fork:1
```

Figure 4: Parameters of experimenting with model 7.1.

and particular results are shown in Fig 5.

```
   16142.1289
   16054.8332
2: 15908.9637 this is the optimum
   16256.8556
   16164.8268      CPU time in seconds:
   16340.3537       2.51
```

Figure 5: Output of experiment with model 7.1

Both types of experiments show the possibility of a wider set of experiments. In future we plan to provide a systematic set of experiments with stepwise change of parameters in order to get an overal view of dependency between parameters and results.

8 Summary

A service system is often viewed as anticipatory one (in a weak sense), because every day one has to anticipate its operation and accordingly formulate scheduling in it. Often the scheduling is made by primitive methods: the formal model figuring in the anticipatory characterization of the system is reduced to several simple formulas.

Simulation of the development that would be caused particular variants is a much better model, as it can reflect any details of the possible future influences and as it finally admits to determine the most efficient one of the variants. The configurations of the service systems for a large spectrum and the limits for scheduling given by physical properties of those system as well; so the object-oriented programming is a good technique for implementation of the simulation models respecting that large spectrums.

A (human) system that designs the initial state of a service system (among other, the number of workers, machines and transport tools, and – in certain cases – their configuration in space) is also an anticipatory system, because it should anticipate the consequences of the design to last a longer time and according to it decide which should be the optimal initial configuration of the designed system.

Simulation models are extremely suitable ones also for this purpose and the object-oriented programming is of the same high quality paradigm as in the construction of the everyday anticipation models. Nevertheless, in that case of simulation, the (other) simulation in everyday anticipation has to be included in the model, because otherwise the model would give non-realistic data (a general proof of that statement is in [6] and [9]). Such "nesting" of a simulation model into another one is a difficult task. The authors were successful using programming language that offers a full synthesis of object orientation, process orientation and block orientation. They used SIMULA [7], [8], which is one of a few languages with these three orientations; moreover, its implementation for PC is free and efficient in computing time and memory space.

Abstract idea of a community of eating Chinese philosophers is a good (and namely widely known) bench mark for testing the proposed run of the service systems. It was simulated by the same manner as the real service systems.

In future, we plan to use several strategies, several fitness functions and preferences in group decision and apply inductive methods from [11].

References

1. Rosen Robert (1985). Anticipatory Systems. Pergamon Press.
2. 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.
3. Dahl Ole-Johan (1964). Discrete Event Simulation Languages. Norwegian Computing Center, Oslo. Reprinted in [4]
4. Genuys Fernand, editor (1968). Programming Languages. Academic Press.
5. Chochol Stefan and Kindler Eugene (1983) Simulation of veterinary sanitation. Simulation of Systems'83. Published by Czechoslovak Society for Science and Technology, pp. 173-176.
6. Kindler Eugene (1999) Simulation of Systems That Contain Simulating Elements. Atti della Conferenza Annuale della Italian Society for Computer Simulation, 15. Giugno 1999. Published by Italian Society for Computer Simulation), pp. 103-108.

7. Dahl Ole-Johan, Myhrhaug Bjorn, and Nygaard Kristen (1968). Common Base Language Norsk Regnesentralen, Oslo. 2nd edition 1972, 3rd edition 1982, 4th edition 1984.
8. SIMULA Standard (1989). SIMULA a.s., Oslo.
9. Kindler Eugene (2000) Chance for SIMULA. ASU Newsletter, Vol. 26, no. 1, pp. 2-26
10. Kindler Eugene (2006): Object-Oriented Representations of Formal Theories as Tools for Simulation of Anticipatory Systems. Computing Anticipatory Systems: CASYS 2005 – Seventh International Conference. Edited by Daniel M. Dubois, Published by The American Institute of Physics, AIP Conference Proceedings 839, pp. 253-259.
11. Horváth, Tomáš and Vojtáš Petr (2006) Ordinal Classification with Monotonicity Constraints. ICDM 2006 – 6th Industrial Conference on Data Mining. Edited by Petra Perner, Published by Springer, Lecture Notes of Artificial Inteligence 4065, pp. 217-225.