Modelling Parallel Service Systems in GPSS

Marko Žibert[1], Miroljub Kljajić[2]

[1]ZZZS, Miklošičeva 24, 1000 Ljubljana, Slovenia, marko.zibert@zzzs.si
[2]Univerza v Mariboru, Fakulteta za organizacijske vede, 4000 Kranj, Kidričeva 55a, Slovenia, miroljub.kljajic@fov.uni-mb.si

This article treats parallel service system modelling in the GPSS simulation language. The transactions entering such systems select between numerous different servers and we can mostly detect two rules in the selecting of the appropriate server. The first rule always gives the first few (regarding its position in the system) entities (either servers or queues) precedence over the others, while the second rule always treats all the equal entities evenly and selects among them quite randomly. Since GPSS normally operates by the first rule, we frequently come up against difficulties when modelling systems that serve by another rule. The present article offers a methodology how to solve this problem within GPSS.

Key words: discrete simulation, modelling, GPSS, parallel service systems, queuing theory

Modeliranje sistemov paralelne strežbe v GPSS-u

Članek obravnava modeliranje paralelnih strežnih sistemov v simulacijskem jeziku GPSS. Transakcije, ki vstopajo v takšne sisteme, izbirajo med večjim številom strežnih mest. Pri zasedanju teh mest pa lahko v grobem zasledimo dva različna pravila. Prvo pravilo daje prednost zasedanju prvih (po svoji poziciji v sistemu) entitet (bodisi strežnikov, bodisi čakalnih vrst), medtem ko drugo pravilo obravnava te entitete enakovredno in izbira med njimi povsem naključno. Ker GPSS v svojem delovanju privzema prvo pravilo, lahko pri modeliranju sistemov, ki strežejo po drugem pravilu, pogosto naletimo na določene težave. Pričujoči prispevek ponuja metodologijo, kako znotraj tega jezika reševati omenjeni problem.

Ključne besede: diskretna simulacija, modeliranje, GPSS, sistemi paralelne strežbe, teorija vrst

1 Definition of the Problem

The parallel service of complex systems is currently an increasingly important research area. This area is gaining in significance as computer science progresses and a lot of scientific periodicals and reviews are now occupied with this field of studies (Katwijk and Zalewski, 1999). Namely the most common problem in service systems recently is the increasing demand for processing a large volume of transactions in real time. These requests could be normally complied with by simply decomposing the original system and its base activity into more dependent subsystems, each with its own activity. But by doing this, new problems can turn up. One of them is the distribution or allocation of the incoming transaction evenly to all the subsystems (the problem of load balancing). The problem can be solved using a variety of theoretical approaches, for instance by intelligent agents (Wooldridge and Jennings, 1995), by Markov chains (Rosenthal, 2000; Song et al., 2004), by Petri nets (Murata, 1989) and by the simulation method (Guariso et al., 1996). In this article, the simulation method, carried out by digital computer, is being used (the computer simulation method).

For the necessity of the simulation and the modelling, a lot of simulation languages (compilers as well as interpreters) have now been developed (Sang et al., 1994). They are being executed on various computers and on the different types of operation systems. One of the first of these languages, and at the same time also the most common, is the GPSS language (General Purpose Simulation System), which was developed in the early sixties for analyzing the responses of the IBM mainframe systems (Blake and Gordon, 1964). At that time it was called General Purpose Systems Simulator (Gordon, 1962). The main GPSS emphasized characteristics (Crain, 1997; Crain, 1998; Crain and Henriksen, 1999; Henriksen and Crain, 2000) that made it very popular among the end-users, such as:

- It was developed for different computer environments (IBM 370 mainframes, personal computers, etc)
- Different versions of GPSS are executable under different operation systems (Multiple Virtual Storage – GPSSSV, Disk Operating System – GPSS/PC)
- The base components of the simulation language (blocks) represent the constituents of the system very well, so we can quickly and easily model any service system taken from reality.
- It creates precise default statistics and reports during the execution of the simulation.
- It is able to perform additional statistics and reports on request.
- Through the **HELP** block it can access an external user-written program (in FORTRAN).

One of the most important characteristics listed above is certainly the structure of the simulation language.

Its main components, semantically meaningful model building blocks, are trying to functionally imitate a particular constituent part of the serving system. So the block names, such as **ADVANCE**, **ASSEMBLE**, **ENTER**, **LEAVE**, **RELEASE**, **SEIZE**, **TEST**, **TRANSFER**, **QUEUE** etc., allow even the uninitiated user to follow the logical flow of a model, at least roughly (Chisman, 1992). In fact, these blocks are just more or less adequate computer projections of the functioning constituents. Thus, without much knowledge of programming and by simply arranging these blocks as they can be seen in reality, we can quickly and easily build precise computer model of the real world system.

In spite of the fact that GPSS is a very user friendly simulation tool, users are not always successful in their modelling of reality. Although in some cases the simulation model is properly built according to the modelling methodology rules (and is also submitted to the syntax rules of GPSS) some considerable discrepancies between the behaviour of the model and the real system can be noticed during the phase of the model evaluation and validation.

The discrepancies described are particularly visible when the simulated system has more equal parallel servers and each of them has the same service characteristics. This means that the service times of each server have the same mean, the same variance and the same statistical distribution. In most cases, as we can also expect, the workload in such systems is evenly distributed among all of the servers. However, the GPSS simulation model that ought to represent such a system, contrary to our expectation, shows unequally loaded servers. In other words, the results of the simulation always indicates that the utilization is the highest at the first server and then it gradually decreases. If the occupation rate per server in the model – the utilization rate that is defined as the fraction of the time the server is working (Adan and Resing, 2001) – increases then the differences in the workloads among particular servers lessen, but the declining trend of the server utilization (from the first server to the last) still exists.

Considering this declining trend, it can be concluded that the discrepancy (the deviation from reality) is especially notable when the modelled systems have more parallel servers than they really need on behalf of system reliability and availability. Under normal circumstances most of these servers would simply be redundant, but in the area of informatics we are frequently dealing with automatic server systems that must be firmly reliable and continuously available, sometimes even under conditions of emergency and under minimum control by the operator. These requirements can be easily complied with some additional parallel servers that could normally be spared.

In this way (by adding additional parallel servers to the system) we are, of course, decreasing the occupation rate per server and, as was said before, we are also increasing the unsuitability of the GPSS model by contrast with the real system. Such a model usually shows that only first few servers are somewhat utilized while the others are completely free and standing idle.

The reasons for the problem described are in special GPSS blocks – the **TRANSFER** and **SELECT** blocks – designed for routing transactions to the target server. Various attributes of some sequential permanent entities, such as **facility** and **queue,** are compared in these blocks. The compared attribute of the **facility** entity is its current state of occupation (whether it is busy or not) and the compared attribute of the **queue** entity is the current length of the queue (the number of transactions waiting in the queue). If these compared attributes are equal then the current transaction always picks out the first positioned feasible entity (in GPSS programme code). For example, if the first n parallel servers in a model are occupied and if the next servers from n+1 to n+k are free, in this case the GPSS simulation always chooses the (n+1)-th server to execute the current transaction.

In reality the server systems more often than not behave quite differently under these circumstances. When the attributes of the compared entities are equal then one of the suitable entity is chosen by transaction clearly at random in most cases. We can experience this especially in the area of informatics where the randomness is even coded into the programmes, subroutines, macros, distribution modules etc. (Cicsplex SM Concepts and Planning; Žibert, 2005). So in the above case the transaction wouldn't precisely pick out the (n+1)-th server but, on the contrary, it would select any among the free k servers (from n+1 to n+k).

Although the server systems with the characteristics described are not very numerous, they can still be found in the real world. Mostly they are connected with the single queue that leads to the very first service facility. All the other service facilities are arranged in a row, one after another at some proper physical distance to each other (this discipline can be often carried out in banks where the customers join a single queue and the first person in line physically engages the nearest free bank-teller), so the transaction (the client, customer, etc.), after leaving the single queue, always seizes its nearest server.

Regarding our brief outline of the activities in the parallel server systems, we can conclude that the real issue is the order in which transactions seize one entity among all the equivalent entities (in case that the entity is a server facility), or enter one entity among all the equivalent entities (in case that the entity is a queue). Although there are also some other possibilities from the real world – especially where people (customers), with their characteristic behaviour, represent the transactions in a system (Azar et al., 1994; Mitzenmacher, 1997) – we would stress that in both cases the transaction serving could be:

- in random order, or in disorder (which is more frequent, even standard in some cases – and we could name it as service in random order);
- in an order of precedence (which is not very common in the real world but it is always used in the modelling with the GPSS programming language – which we could name the service in order of precedence).

As a result of the approaches explained, we can state that GPSS modelling of the parallel server system with the service in order of precedence is very easy and uncomplicated. Namely, both the system itself and the GPSS model use the service in order of precedence, so the simulation results are usually in accordance with what happens in the real system.

We always come up to against difficulties, on the

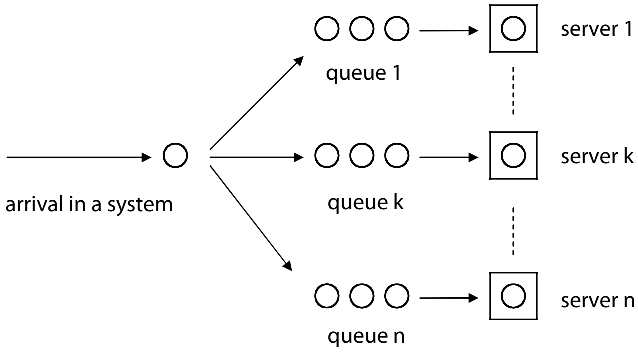

Figure 1: The scheme of the multiple servers system, each server with its own waiting queue

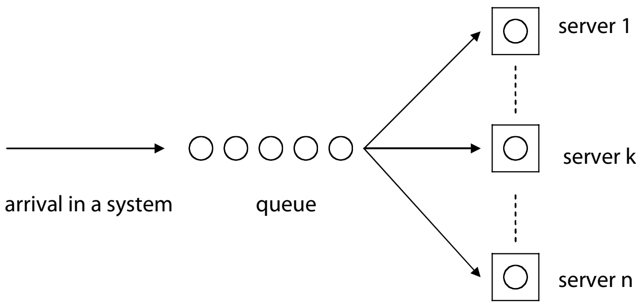

Figure 2: The scheme of the multiple server system with a single waiting queue

other hand, when we try to build a GPSS model of a parallel server system with the service in random order. The reason is obvious because the system and its model use different types of service order. As we said earlier the modelling problems are even bigger when the occupation rate per server in the system is low. In this case the simulated utilization of the parallel servers in the GPSS model would be completely inadequate.

That is why, in the following chapters, we are trying to develop a new GPSS methodology of modelling and simulating parallel server systems with the service in random order.

2 The Two Main Types of Parallel Service

The existing GPSS methodology of modelling parallel server systems depends on the general type of the parallel service we want to simulate. We can distinguish two main types of parallel service and each of them has its special solution within the normal (classical) usage of the GPSS programming language. That means that it has its own sequence of various GPSS blocks that should illustrate the functioning of the system.

The term "sequence of blocks" is not something that is fixed and defined once for all by the GPSS developers. We should consider it just as one variation among the many possibilities that GPSS programmer can normally use. The stress here is not just on the "sequence of blocks" but also on the normal or classical usage of the GPSS language. Generally speaking we have two main "sequences of blocks" in classical GPSS programming for depicting parallel server systems. Although there are certainly many individual variations in recording these blocks, they are almost always based on either the **SELECT MIN** or the **TRANSFER ALL** structure.

These two main types are:

- multiple servers, each with its own waiting queue (Figure 1)
- multiple servers, all with one single waiting queue (Figure 2)

At first sight it seems that both systems are very complex and thus hard to model in the GPSS language. It seems that by attempting this we couldn't avoid many pages of long block sequences contributing to almost completely unclear, complicated and messy programme code. But the boot is on other foot. Lots of tough work and tiresome coding can be saved by simply using indirect addressing (Chisman, 1992). But on the other hand, by using indirect addressing we also loose something. The visual flow of transactions through the system becomes clouded and confused.

The following programme codes show us the possibilities of how to use indirect addressing in classical programming for such parallel systems. Figure 3 represents the GPSS model of a multiple server system where each server has its own waiting queue, while Figure 4 shows a model of a similar system with a single waiting queue.

3 The Graphic Representation of the Problem

If we tried to persuasively demonstrate the functioning of the GPSS models presented in Figure 3 and Figure 4 we would have to establish some requirements first, namely:

- the number of parallel servers in the system
- the service time distribution of each server
- the distribution of transaction time between arrivals into the system.

We can also use data from the real world for the purpose of our research, especially from the computer world. So for the time between the arrivals function and for the service time function we can use the tables published in (Žibert, 1999). For the sake of simplicity let us also assume that all the servers are equivalent in our model (meaning that the service time function is the same for all the parallel servers in the whole system). In this way Figure 5 and Figure 6 show us complemented and developed programmes (based originally on Figure 3 and Figure 4).

```
    REALLOCATE COM,32720
    SIMULATE
*
*   Parallel server system - n servers each with its own waiting queue.
*
SERVER1                 EQU         1,F
SERVER.                 EQU         .,F
SERVERN                 EQU         N,F     n servers
QUEUE1                  EQU         1,Q
QUEUE.                  EQU         .,Q
QUEUEN                  EQU         N,Q     n waiting queues
PROCES1                 EQU         1,Z
PROCES.                 EQU         .,Z
PROCESN                 EQU         N,Z     n process functions
PROCES1                 FUNCTION    RNx,Cx Service time distribution 1
          0,../1,..
PROCES.                 FUNCTION    RNx,Cx Service time distribution .
          0,../1,..
PROCESN                 FUNCTION    RNx,Cx Service time distribution N
          0,../1,..
PRIHOD                  FUNCTION    RNx,Cx Time between arrivals distribution
          0,../1,..
                        GENERATE    FN$PRIHOD
*                       The TRANSFER block determines which entity in the range
                        from
*                       QUEUE1 to QUEUEN has the minimum content and then places
                        the
*                       number of this entity into parameter 1.

                        SELECT MIN  1,QUEUE1,QUEUEN,,Q
                        QUEUE       P1
                        SEIZE       P1
                        DEPART      P1
                        ADVANCE     FN*P1
                        RELEASE     P1
                        TERMINATE   1
*
                        START       xxx     The number of processed transactions
                        END
```

Figure 3: Classical usage of GPSS blocks for modelling a system of n servers, each with its own waiting queue

```
    REALLOCATE COM,32720
    SIMULATE
*
*   Parallel server system - n servers with one single waiting queue.
*
SERVER1                 EQU         1,F
SERVER.                 EQU         .,F
SERVERN                 EQU         N,F
QUEUE1                  EQU         1,Q
PROCES1                 EQU         1,Z
PROCES.                 EQU         .,Z
PROCESN                 EQU         N,Z
PROCES1                 FUNCTION    RNx,Cx Service time distribution 1
          0,../1,..
PROCES.                 FUNCTION    RNx,Cx Service time distribution .
          0,../1,..
PROCESN                 FUNCTION    RNx,Cx Service time distribution N
          0,../1,..
PRIHOD                  FUNCTION    RNx,Cx Time between arrivals distribution
          0,../1,..
*
                        GENERATE    FN$PRIHOD
                        QUEUE       QUEUE1
*The TRANSFER block will see if the engaging transaction can go to the first
*location (BCPU1); if not, it will try to go to the next (BCPU2);if not, then to
*(BCPU3), until it tries the last location (BCPUN). If it cannot send it anywhere,
*it starts all over again, until it can finally move transaction to one of these
*locations.
                        TRANSFER    ALL,BCPU1,BCPUN,3
BCPU1                   SEIZE       1
                        ASSIGN      1,1
                        TRANSFER    ,DALJE
BCPU2                   SEIZE       2
                        ASSIGN      1,2
                        TRANSFER    ,DALJE
BCPU.                   SEIZE       .
                        ASSIGN      1,.
                        TRANSFER    ,DALJE
BCPUN                   SEIZE       N
                        ASSIGN      1,N
DALJE                   DEPART      QUEUE1
                        ADVANCE     FN*P1
                        RELEASE     P1
                        TERMINATE   1
*
                        START       xxx    The number of processed transactions
                        END
```

Figure 4: Classical usage of GPSS blocks for modelling a system of n servers with one single waiting queue

```
    REALLOCATE COM,32720
    SIMULATE
*
*   Parallel server system - n servers each with its own waiting queue.
*
SERVER1             EQU             1,F
SERVER.             EQU             .,F
SERVERN             EQU             N,F
QUEUE1              EQU             1,Q
QUEUE.              EQU             .,Q
QUEUEN              EQU             N,Q
PROCES1             EQU             1,Z
PROCES.             EQU             .,Z
PROCESN             EQU             N,Z
PROCES1             FUNCTION        RN1,C16         Service time distribution 1
                                                    (in seconds)
     0.0000,0.0/0.3867,0.1/0.5693,0.2/0.6829,0.3/0.7604,0.4/0.8117,0.5/
     0.8463,0.6/0.8702,0.7/0.8887,0.8/0.9036,0.9/0.9150,1.0/0.9319,1.2/
     0.9476,1.5/0.9648,2.0/0.9795,3.0/1.0000,5.0

PROCES.             FUNCTION        RN1,C16         Service time distribution .
                                                    (in seconds)
* The same data as above in PROCES1

PROCESN             FUNCTION        RN1,C16         Service time distribution N
                                                    (in seconds)
* The same data as above in PROCES1

FPRIHOD             FUNCTION        AC1,C62         Time between arrivals
                                                    distribution (10 hours)
* The same data as above in PROCES1

VPRIHOD             FVARIABLE       FN$FPRIHOD*(ABS(LOG(1-(RN2/1000))))
                    GENERATE        V$VPRIHOD,,ST   The simulation begins
                                                    at time = ST
                    SELECT MIN      1,QUEUE1,QUEUEN,,Q
                    QUEUE           P1
                    SEIZE           P1
                    DEPART          P1
                    ADVANCE         FN*P1
                    RELEASE         P1
                    TERMINATE
*
                    GENERATE        DT                The simulation lasts DT
                                                      seconds
                    TERMINATE       1
*
                    START           1
                    END
```

Figure 5: The classical model of a system with n servers and n waiting queues that processes statistical data from (Žibert, 1999)

```
    REALLOCATE COM,32720
    SIMULATE
*
*   Parallel server system - n servers with one single waiting queue.
*
SERVER1        EQU          1,F
SERVER.        EQU          .,F
SERVERN        EQU          N,F
QUEUE1         EQU          1,Q
PROCES1        EQU          1,Z
PROCES.        EQU          .,Z
PROCESN        EQU          N,Z
PROCES1        FUNCTION     RN1,C16              Service time distribution 1
                                                 (in seconds)
     0.0000,0.0/0.3867,0.1/0.5693,0.2/0.6829,0.3/0.7604,0.4/0.8117,0.5/
     0.8463,0.6/0.8702,0.7/0.8887,0.8/0.9036,0.9/0.9150,1.0/0.9319,1.2/
     0.9476,1.5/0.9648,2.0/0.9795,3.0/1.0000,5.0
PROCES.        FUNCTION     RN1,C16              Service time distribution .
                                                 (in seconds)
* The same data as above in PROCES1
PROCESN        FUNCTION     RN1,C16              Service time distribution N
                                                 (in seconds)
* The same data as above in PROCES1
FPRIHOD  FUNCTION  AC1,C62 Time between arrivals distribution (10 hours)
* Data for this function are defined in FPRIHOD in Figure 3
VPRIHOD        FVARIABLE    FN$FPRIHOD*(ABS(LOG(1-(RN2/1000))))
               GENERATE     V$VPRIHOD,,ST        The simulation begins at time = ST
               QUEUE        QUEUE1
               TRANSFER     ALL,BCPU1,BCPUN,3
BCPU1          SEIZE        1
               ASSIGN       1,1
               TRANSFER     ,DALJE
BCPU2          SEIZE        2
               ASSIGN       1,2
               TRANSFER     ,DALJE
BCPU.          SEIZE        .
               ASSIGN       1,.
               TRANSFER     ,DALJE
BCPUN          SEIZE        N
               ASSIGN       1,N
DALJE          DEPART       QUEUE1
               ADVANCE      FN*P1
               RELEASE      P1
               TERMINATE
*
               GENERATE     DT                   The simulation lasts DT seconds
               TERMINATE    1
*
               START        1
               END
```

Figure 6: The classical model of a system with n servers and a single waiting queue that processes statistical data from (Žibert, 1999)

Table 1: Average server utilization (column 3) and average queue content (column 5) depending on the number of servers in the model (column 1)

Number of servers	Average server utilization		Average queue content	
3	SERVER1	0.924	QUEUE1	2.712
	SERVER2	0.832	QUEUE2	2.388
	SERVER3	0.697	QUEUE3	2.143
4	SERVER1	0.874	QUEUE1	0.932
	SERVER2	0.740	QUEUE2	0.738
	SERVER3	0.529	QUEUE3	0.506
	SERVER4	0.310	QUEUE4	0.307
5	SERVER1	0.865	QUEUE1	0.705
	SERVER2	0.722	QUEUE2	0.527
	SERVER3	0.497	QUEUE3	0.325
	SERVER4	0.259	QUEUE4	0.158
	SERVER5	0.110	QUEUE5	0.065
6	SERVER1	0.865	QUEUE1	0.657
	SERVER2	0.713	QUEUE2	0.495
	SERVER3	0.495	QUEUE3	0.308
	SERVER4	0.254	QUEUE4	0.144
	SERVER5	0.097	QUEUE5	0.044
	SERVER6	0.030	QUEUE6	0.012

Table 2: The number of transactions (column 3) and their percentages (column 4) passed through the individual servers (column 2) in the model with N (column 1) parallel servers

Number of servers	Server	Number of transactions	Percentage [%]
3	SERVER1	8.587	0.378
	SERVER2	7.729	0.341
	SERVER3	6.366	0.281
	SUM	22.682	1.000
4	SERVER1	8.189	0.361
	SERVER2	6.658	0.294
	SERVER3	4.887	0.215
	SERVER4	2.949	0.130
	SUM	22.683	1.000
5	SERVER1	8.120	0.358
	SERVER2	6.372	0.281
	SERVER3	4.625	0.204
	SERVER4	2.501	0.110
	SERVER5	1.065	0.047
	SUM	22.683	1.000
6	SERVER1	8.032	0.355
	SERVER2	6.430	0.283
	SERVER3	4.516	0.199
	SERVER4	2.436	0.107
	SERVER5	999	0.044
	SERVER6	270	0.012
	SUM	22.683	1.000

Having enhanced our models with real data, we are now able to carry out the series of simulations. In each simulation we can apply some modifications, such as the number of concurrent servers (N), the starting time (defined by letter Z in our programme), the duration of the simulation (defined by letter Y in our programme) etc.

First we can try using the model from Figure 5. For testing purposes we can accept the following parameters:

- the starting time is zero (ST = 0)
- the duration time is one hour (DT = 3600)
- the number of servers is increasing from the minimum to the maximum reasonable number (meaning that there are at least certain number of servers with the attention of avoiding queues that are too long in the model and that all N servers in our model have some traffic -min. <= N =< max.).

The results are represented in Table 1.

As we can see from the table above, the utilization of the servers in the first few positions in the programme code (SERVER1 and SERVER2) is quite high – considerably higher in comparison with the servers positioned at the end of the code (SERVER4, SERVER5 and SERVER6). Furthermore, we can't fail to observe that the utilization of these same servers (SERVER1 and SERVER2) is not changed much by adding additional servers in the model. This can also be seen by looking at the number of transactions (and their percentages) passed through the individual servers (Table 2).

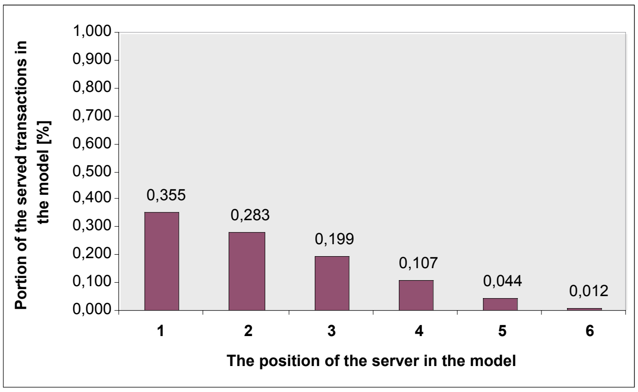

Figure 7: The distribution of the service in the classical GPSS model with six parallel servers

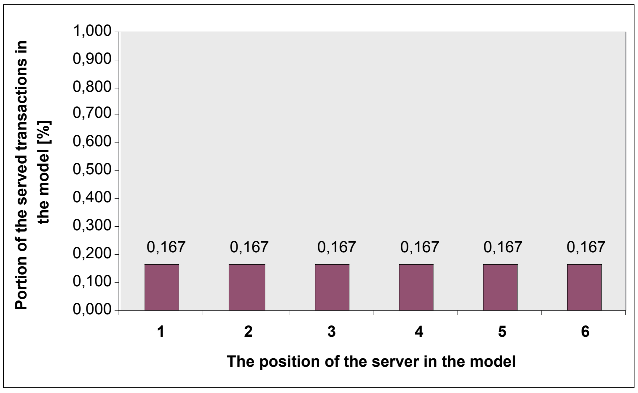

Figure 8: The expected distribution of the service in the classical GPSS model with 6 servers

The same can be seen more obviously in a graphic way, especially for the model with six parallel servers (Figure 7). Here we can clearly observe the declining trend of the server utilization. Thus, the first server in the model executes almost 36 percent of all the completed transactions and the sixth server executes only one and if we expanded our model by adding some new servers then they would be completely idle.

Of course, considering the service in random order, which was our presumption, we would expect that the above graph would be quite different and similar to Figure 8.

But what would we get if the traffic in the model (the number of entering transactions) diminished rapidly? Let's now change our GPSS programme from Figure 6 (the model with n parallel servers and with a single waiting queue) as follows:

- the starting time ST = 33000 (though the density of the transaction arrivals is much lower)
- the duration time is again one hour, DT = 3600
- the number of server is increased from 2 to 6 (2 <= N =< 6).

Table 3: The number of transactions (column 3) and their percentages (column 4) that passed through the individual servers (column 2) in the model with N parallel servers (column 1) during conditions of low traffic density

Number of servers	Server	Number of transactions	Percentage [%]
2	SERVER1	449	0.947
	SERVER	025	0.053
	SUM	474	1.000
3	SERVER1	449	0.947
	SERVER2	025	0.053
	SERVER3	000	0.000
	SUM	474	1.000
4	SERVER1	449	0.947
	SERVER2	025	0.053
	SERVER3	000	0.000
	SERVER4	000	0.000
	SUM	474	1.000
5	SERVER1	449	0.947
	SERVER2	025	0.053
	SERVER3	000	0.000
	SERVER4	000	0.000
	SERVER5	000	0.000
	SUM	474	1.000
6	SERVER1	449	0.947
	SERVER2	025	0.053
	SERVER3	000	0.000
	SERVER4	000	0.000
	SERVER5	000	0.000
	SERVER6	000	0.000
	SUM	474	1.000

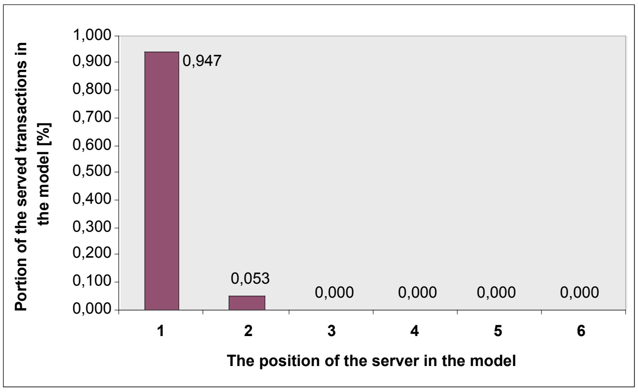

Figure 9: The distribution of the service in the classical GPSS model (with six parallel servers) on condition of low traffic density

The results are presented in Table 3 and in Figure 9 for the model with six servers.

Straight away we can see that the model shows quite unrealistic situation during conditions of low traffic density (and by that also a low occupation rate per server). Barely more than one server is utilized in the model (i. e. the first one). In our case it is (only by chance) fully loaded while all the other servers are practically unattached. This example clearly demonstrates the discrepancy of the model and its dependence on the occupation rate per server explained earlier. Hence it follows that each and every possible solution should be proved under the same conditions – i.e. models with a low occupation rate per server.

4 The Solution of the Problem

In the previous chapters we established and proved that classical (ordinary) usage of GPSS blocks in modelling doesn't take into consideration the principle of service in random order. So whenever we model a system in GPSS that operates in this way we always come up against difficulties. However, in spite of everything, this principle can be achieved. Taking into account that there are two main types of parallel service (described in Figure 1 and Figure 2) we will also offer two different solutions for each type.

For the first type (the systems containing n servers each with its own queue) this difficult task could be tackled in the following way. At first the GPSS programme determines the length of the shortest queue in the system (**SELECT MIN**). Then it randomly (variable **VAR1**) chooses one of the feasible queues (**ASSIGN**) and compares its length with the length of the shortest one (**TEST E**). If both lengths are equal then the transaction is normally sent to the randomly chosen queue (**QUEUE**). Otherwise the programme picks out another waiting queue (the execution of the programme returns to label "**PONOVNO**"). Figure 10 shows the principle part of the GPSS programme explained above.

Our sample programme as a whole, upgraded using the method described, would look like that shown in Figure 11.

```
VAR1        VARIABLE     ((RN3*N/1000)+1)
            GENERATE     ....
            SELECT MIN   1,QUEUE1,QUEUEN,,Q
PONOVNO     ASSIGN       2,V$VAR1
            TEST         E                  Q*P1,Q*P2,PONOVNO
            QUEUE                           P2
```

Figure 10: The section of the GPSS programme that solves the problem in a model with n servers and n queues

```
    REALLOCATE COM,32720
    SIMULATE
*
*   Parallel server system - n servers each with its own waiting queue.
*   The upgraded variant
*
SERVER1       EQU           1,F
SERVER.       EQU           .,F
SERVERN       EQU           N,F
QUEUE1        EQU           1,Q
QUEUE.        EQU           .,Q
QUEUEN        EQU           N,Q
PROCES1       EQU           1,Z
PROCES.       EQU           .,Z
PROCESN       EQU           N,Z
PROCES1       FUNCTION      RN1,C16             Service time distribution 1 (in seconds)
  0.0000,0.0/0.3867,0.1/0.5693,0.2/0.6829,0.3/0.7604,0.4/0.8117,0.5/
  0.8463,0.6/0.8702,0.7/0.8887,0.8/0.9036,0.9/0.9150,1.0/0.9319,1.2/
  0.9476,1.5/0.9648,2.0/0.9795,3.0/1.0000,5.0
PROCES.       FUNCTION      RN1,C16             Service time distribution . (in seconds)
* The same data as above in PROCES1
PROCESN       FUNCTION      RN1,C16             Service time distribution N (in seconds)
* The same data as above in PROCES1
FPRIHOD       FUNCTION      AC1,C62             Time between arrivals distribution (10 hours)
* Data for this function are defined in FPRIHOD in Figure 3
VPRIHOD       FVARIABLE     FN$FPRIHOD*(ABS(LOG(1-(RN2/1000))))
              INITIAL       X$SERVNUM,N         Number of servers = N
VAR1          VARIABLE      ((RN3*X$SERVNUM/1000)+1)  A randomly chosen queue
              GENERATE      V$VPRIHOD,,ST       The simulation begins at time = ST
              SELECT MIN    1,QUEUE1,QUEUEN,,Q
PONOVNO       ASSIGN        2,V$VAR1
              TEST  E       Q*P1,Q*P2,PONOVNO
              QUEUE         P2
              SEIZE         P2
              DEPART        P2
              ADVANCE       FN*P2
              RELEASE       P2
              TERMINATE
*
              GENERATE      DT                  The simulation lasts DT seconds
              TERMINATE     1
*
              START         1
              END
```

Figure 11: Our upgraded model of a system with n servers and n waiting queues that processes the same statistical data as the programme in Figure 5

Table 4: The number of transactions (column 3) and their percentages (column 4) passed through the individual servers (column 2) in the upgraded model with N (column 1) parallel servers

Number of servers	Server	Number of transactions	Percentage [%]
2	SERVER1	225	0.475
	SERVER	249	0.525
	SUM	474	1.000
3	SERVER1	152	0.321
	SERVER2	154	0.325
	SERVER3	168	0.354
	SUM	474	1.000
4	SERVER1	114	0.241
	SERVER2	111	0.234
	SERVER3	121	0.255
	SERVER4	128	0.270
	SUM	474	1.000
5	SERVER1	84	0.177
	SERVER2	96	0.203
	SERVER3	95	0.200
	SERVER4	93	0.196
	SERVER5	106	0.224
	SUM	474	1.000
6	SERVER1	71	0.150
	SERVER2	81	0.171
	SERVER3	73	0.154
	SERVER4	81	0.171
	SERVER5	78	0.164
	SERVER6	90	0.190
	SUM	474	1.000

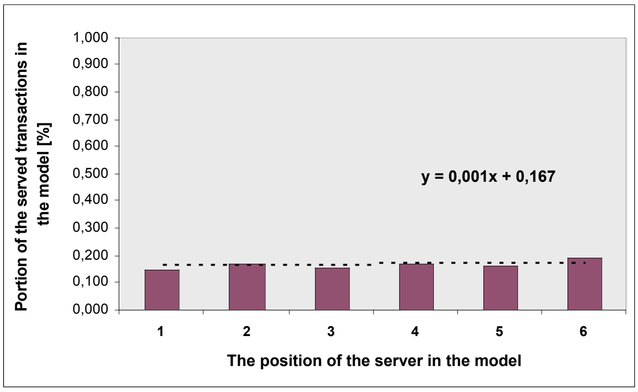

Figure 12: The distribution of the service in the upgraded GPSS model with a dotted trend line

When the upgraded GPSS programme is executed under the same conditions as before (the starting time ST = 33000, the duration time DT = 3600 and the number of servers ranging between 2 and 6) we get the following simulation results (Table 4) and the following graph for a model with six servers (Figure 12).

Right away we can recognize that the behaviour of the model is quite different to that in all the earlier cases. As we can see, each server now seems to complete approximately the same percentage of the incoming transactions so the workload in our upgraded model quite realistically seems to be evenly distributed among all of the servers in the system. That hypothesis was even statistically confirmed using the chi-squared test in (Žibert, 2005).

For the parallel service model using a single waiting queue (earlier defined as the second type) it is much harder to find a solution. The classic GPSS system uses a long sequence of blocks for this purpose. Thus in our sample we controlled the flow of a transaction within the blocks declared in the transfer block (TRANSFER ALL, BCPU1, BCPUN, 3). That means that we controlled it all the way from the **TRANSFER ALL** block to the block labelled **BCPUN** plus three additional subsequent blocks. It seems that all the blocks in between form an indivisible entity where randomness of any kind can not be taken into account.

```
INITIAL      X$CPUNUM,N                          Number of servers = N
VAR1         VARIABLE      ((RN3*X$CPUNUM/1000)+1)
             GENERATE      ...
             QUEUE         QUEUE1
             ASSIGN        1,X$CPUNUM-1
             ASSIGN        2,V$VAR1
PONOVNO      TEST L        P(X$CPUNUM-P1+1),X$CPUNUM,ZACETEK
             ASSIGN        (X$CPUNUM-P1+2),P(X$CPUNUM-P1+1)+1
             TRANSFER      ,ZANKA
ZACETEK      ASSIGN        (X$CPUNUM-P1+2),1
             TRANSFER      ,ZANKA
ZANKA        LOOP          1,PONOVNO
```

Figure 13: The complex section of the GPSS programme that solves the problem in a model with n servers and a single queue

However, the problem here can be also grappled with. By applying indirect addressing in the **SEIZE** blocks we could always use one of the transaction parameters (**P1**, **P2**, **P3**, etc). That means that the transaction occupies the facility that is coded in that parameter. Thus, if we changed the contents of all those parameters belonging to the transaction at the time of its birth (generation), we would

```
REALLOCATE    COM,32720
    SIMULATE
*
*   Parallel server system - n servers with a single waiting queue.
*   The upgraded variant
*
SERVER1       EQU           1,F
SERVER.       EQU           .,F
SERVERN       EQU           N,F
QUEUE1        EQU           1,Q
PROCES1       EQU           1,Z
PROCES.       EQU           .,Z
PROCESN       EQU           N,Z
PROCES1       FUNCTION      RN1,C16    Service time distribution 1 (in seconds)
     0.0000,0.0/0.3867,0.1/0.5693,0.2/0.6829,0.3/0.7604,0.4/0.8117,0.5/
     0.8463,0.6/0.8702,0.7/0.8887,0.8/0.9036,0.9/0.9150,1.0/0.9319,1.2/
     0.9476,1.5/0.9648,2.0/0.9795,3.0/1.0000,5.0
PROCES.       FUNCTION      RN1,C16    Service time distribution . (in seconds)
* The same data as above in PROCES1
PROCESN       FUNCTION      RN1,C16    Service time distribution N (in seconds)
* The same data as above in PROCES1
FPRIHOD       FUNCTION      AC1,C62  Time between arrivals distribution (10 hours)
* Data for this function are defined in FPRIHOD in Figure 3
VPRIHOD       FVARIABLE     FN$FPRIHOD*(ABS(LOG(1-(RN2/1000))))
              INITIAL       X$CPUNUM,N Number of servers = N
VAR1          VARIABLE      ((RN3*X$CPUNUM/1000)+1)   A random number from 1 to N
              GENERATE      V$VPRIHOD,,ST The simulation begins at time = ST
              QUEUE         QUEUE1
*   The start of filling our parameter table
              ASSIGN        1,X$CPUNUM-1
              ASSIGN        2,V$VAR1
PONOVNO       TEST    L     P(X$CPUNUM-P1+1),X$CPUNUM,ZACETEK
              ASSIGN        (X$CPUNUM-P1+2),P(X$CPUNUM-P1+1)+1
              TRANSFER      ,ZANKA
ZACETEK       ASSIGN        (X$CPUNUM-P1+2),1
              TRANSFER      ,ZANKA
ZANKA         LOOP          1,PONOVNO
*   The end of filling our parameter table
              TRANSFER      ALL,BCPU1,BCPUN,5
BCPU1         SEIZE         P2
              DEPART        QUEUE1
              ADVANCE       FN*P2
              RELEASE       P2
              TRANSFER      ,DALJE
BCPU.         SEIZE         P.
              DEPART        QUEUE1
              ADVANCE       FN*P.
              RELEASE       P.
              TRANSFER      ,DALJE
BCPUN         SEIZE         P(N+1)
              DEPART        QUEUE1
              ADVANCE       FN*P(N+1)
              RELEASE       P(N+1)
DALJE                       TERMINATE
*
              GENERATE      DT         The simulation lasts DT seconds
              TERMINATE     1
*
              START         1
              END
```

Figure 14: Our upgraded model of a system with n servers and a single waiting queue that processes the same statistical data as the programme from Figure 6

Table 5: The number of transactions (column 3) and their percentages (column 4) passed through the individual servers (column 2) in the upgraded model with N (column 1) parallel servers

Number of servers	Server	Number of transactions	Percentage [%]
2	SERVER1	226	0.477
	SERVER	248	0.523
	SUM	474	1.000
3	SERVER1	155	0.327
	SERVER2	155	0.327
	SERVER3	164	0.346
	SUM	474	1.000
4	SERVER1	117	0.247
	SERVER2	111	0.234
	SERVER3	122	0.257
	SERVER4	124	0.262
	SUM	474	1.000
5	SERVER1	87	0.183
	SERVER2	96	0.203
	SERVER3	96	0.203
	SERVER4	93	0.196
	SERVER5	102	0.215
	SUM	474	1.000
6	SERVER1	73	0.154
	SERVER2	82	0.173
	SERVER3	73	0.154
	SERVER4	81	0.171
	SERVER5	78	0.165
	SERVER6	87	0.183
	SUM	474	1.000

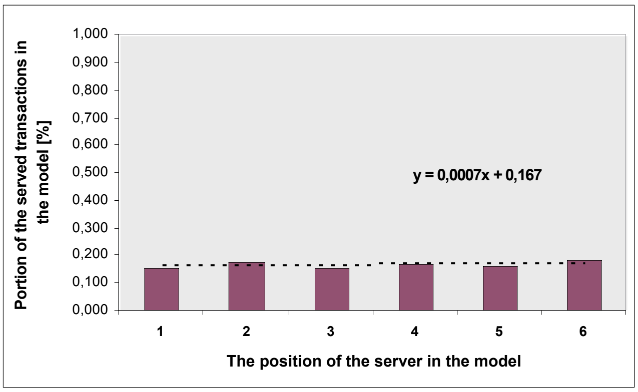

Figure 15: The distribution of the services in the upgraded GPSS model with a dotted trend line

also change all the facilities at hand in the **SEIZE** blocks. All we must do is to create a table of parameters for each generated transaction and fill it randomly with the numbers that represent the appointed facility. Figure 13 shows us how to do it.

This time, our complete sample programme, upgraded using the method described, would look like the that in Figure 14.

The results of the above GPSS simulation model (under the same condition as earlier, with the starting time ST = 33000, the duration time DT = 3600 and the number of servers ranging between 2 and 6) are once again presented as a table (Table 5) and as a graph for a model with six servers (Figure 15).

As before, we can perceive that the servers in the model are treated approximately much the same. So the workload in this upgraded programme could also be considered as evenly distributed among all of the servers (Žibert, 2005).

```
*
*   The definitions of macro IZBIRAQ for systems with n parallel
*   servers and n waiting queues
*
*   Macro parameters:
*   #A – the number of parallel servers – N
*   #B – random number (1 – 9)
*
*   The exit is parameter 2.
*
IZBIRAQ     STARTMACRO  #A,#B
VAR1        VARIABLE    ((#B*(#A+1-P1)/1000)+P1)
            SELECT MIN  1,QUEUE1,#A,,Q
PONOVNO     ASSIGN      2,V$VAR1
            TEST  E     Q*P1,Q*P2,PONOVNO
            ENDMACRO
*
*   The end of macro IZBIRAQ
```

Figure 16: The IZBIRAQ macro for models with n parallel servers and n waiting queues

5 Practical Forms of our Solution

Perhaps it would be helpful for many of us if we also introduced our solution in a rather different form – using so-called macros. This would make the solution more common, even user friendly and (we hope) more applicable. Macros are not just used to be easily and repeatedly called from every possible point inside the programme. They are also applied to shorten large source codes and to make them much easier to understand.

In this way, we present the GPSS macros for the models in Figure 16 and Figure 17.

In next figures (Figure 18 and Figure 19), there are two simple samples of how the above two macros can be used, so the readers can learn by examples.

```
*
*   The definition of macro IZBIRAF for systems with n parallel
*   servers and a single waiting queue
*
*   Macro parameters:
*   #A - the number of parallel servers - N
*   #B - random number (1 - 9)
*
*   The exit is a table of parameters from P2 to PN.
*
IZBIRAF       STARTMACRO    #A,#B
VAR1          VARIABLE      ((#B*#A/1000)+1)
              ASSIGN        1,#A-1
              ASSIGN        2,V$VAR1
PONOVNO       TEST    L     P(#A-P1+1),#A,ZACETEK
              ASSIGN        (#A-P1+2),P(#A-P1+1)+1
              TRANSFER      ,ZANKA
ZACETEK       ASSIGN        (#A-P1+2),1
              TRANSFER      ,ZANKA
ZANKA         LOOP            1,PONOVNO
              ENDMACRO
*
*             The end of macro IZBIRAF
```

Figure 17: The IZBIRAF macro for models with n parallel servers and a single waiting queue

```
    SIMULATE
*
*   System with 5 servers, each with its own waiting queue.
*   The programme calls macro IZBIRAQ.
*
CPU1          EQU           1,F
CPU2          EQU           2,F
CPU3          EQU           3,F
CPU4          EQU           4,F
CPU5          EQU           5,F
QUEUE1        EQU           1,Q
QUEUE2        EQU           2,Q
QUEUE3        EQU           3,Q
QUEUE4        EQU           4,Q
QUEUE5        EQU           5,Q
PROCES1       EQU           1,Z
PROCES2       EQU           2,Z
PROCES3       EQU           3,Z
PROCES4       EQU           4,Z
PROCES5       EQU           5,Z
PROCES1       FUNCTION      RN1,C2
       0,1/1,1
PROCES2       FUNCTION      RN2,C2
       0,1/1,1
PROCES3       FUNCTION      RN3,C2
       0,1/1,1
PROCES4       FUNCTION      RN4,C2
       0,1/1,1
PROCES5       FUNCTION      RN5,C2
       0,1/1,1
              INITIAL       X$CPUNUM,5
*
*             The definition of macro IZBIRAQ
*
IZBIRAQ       STARTMACRO    #A,#B
VAR1          VARIABLE      ((#B*(#A+1-P1)/1000)+P1)
              SELECT MIN    1,QUEUE1,#A,,Q
PONOVNO       ASSIGN        2,V$VAR1
              TEST   E      Q*P1,Q*P2,PONOVNO
              ENDMACRO
*
*             The end of the macro
*
*
*             The main programme
*
              GENERATE      0.6,0.5,,,,27
IZBIRAQ       MACRO         X$CPUNUM,RN6
              QUEUE         P2
              SEIZE         P2
              DEPART        P2
              ADVANCE       FN*P2
              RELEASE       P2
              TERMINATE     1
*
              START         10000
              END
```

Figure 18: A simple programme showing how to use the IZBIRAQ macro

```
    SIMULATE
*
*   System with 3 servers and one waiting queue.
*   The programme calls macro IZBIRAF.
*
SERVER1         EQU             1,F
SERVER2         EQU             2,F
SERVER3         EQU             3,F
QUEUE1          EQU             1,Q
PROCES1         EQU             1,Z
PROCES2         EQU             2,Z
PROCES3         EQU             3,Z
PROCES1         FUNCTION        RN1,C2
       0,1/1,1
PROCES2         FUNCTION        RN2,C2
       0,1/1,1
PROCES3         FUNCTION        RN3,C2
       0,1/1,1
FPRIHOD         FUNCTION  AC1,C7
     00000,1.2/00600,1.4/01200,0.9/01800,0.8/02400,0.75/03000,0.9/03600,1.2
VPRIHOD         FVARIABLE       FN$FPRIHOD*(ABS(LOG(1-(RN2/1000))))
                INITIAL         X$CPUNUM,3
*
*               The start of macro IZBIRAF
*
IZBIRAF         STARTMACRO      #A,#B
VAR1 VARIABLE   ((#B*#A/1000)+1)
                ASSIGN          1,#A-1
                ASSIGN          2,V$VAR1
PONOVNO         TEST   L        P(#A-P1+1),#A,ZACETEK
                ASSIGN          (#A-P1+2),P(#A-P1+1)+1
                TRANSFER        ,ZANKA
ZACETEK         ASSIGN          (#A-P1+2),1
                TRANSFER        ,ZANKA
ZANKA           LOOP              1,PONOVNO
                ENDMACRO
*
*               The end of the macro
*
*               The main programme
*
                GENERATE        V$VPRIHOD
                QUEUE           QUEUE1
IZBIRAF         MACRO           X$CPUNUM,RN4
                TRANSFER        ALL,BCPU1,BCPU3,5
BCPU1           SEIZE           P2
                DEPART          QUEUE1
                ADVANCE         FN*P2
                RELEASE         P2
                TRANSFER        ,DALJE
BCPU2           SEIZE           P3
                DEPART          QUEUE1
                ADVANCE         FN*P3
                RELEASE         P3
                TRANSFER        ,DALJE
BCPU3           SEIZE           P4
                DEPART          QUEUE1
                ADVANCE         FN*P4
                RELEASE         P4
DALJE           TERMINATE
*
                GENERATE        3600
                TERMINATE       1
*
                START           1
                END
```

Figure 19: A simple programme showing how to use the IZBIRAF macro

6 Conclusions

As we said in chapter one when describing the problem, the GPSS modelling of parallel server systems with the service in random order always causes some problems. The final effect of these troubles (and our problem) is more or less presented as a discrepancy between the model and the real system. In some extreme cases the discrepancy could be so large that we are talking about an inadequate model.

In this article we tried to show a methodology of how to surmount the obstacles represented in this simulation language and how to correctly model some prevailing types of parallel service systems. The suggested solution is mainly composed of some additional control statements (as declarations at the beginning of the GPSS programme) and an extra section of block sequence that randomly chooses one of the suitable entities in the model. To make the solution more applicable among users we also went a step further. In this respect, we made it easily available as a macro called from the main programme with some additional parameters.

As presented in the article (especially in the graphs), the solution successfully simulates the behaviour of parallel service systems with the service in random order. Without using it, the model would describe the same system but with different type of service order. In this case it would represent a system with the service in order of precedence, which is immanent to GPSS.

Literature

Adan, I. & Resing, J. (2001). *Queueing Theory*, Department of Mathematics and Computing Science, Eindhoven University of Technology.

Azar, Y., Broder, A., Karlin, A. & Upfal, E. (1999). Balanced Allocations, *SIAM Journal on Computing,* 29(1): 180-200.

Blake, K. & Gordon G. (1964). System simulation with digital computers, *IBM Systems Journal*, **3**(1):14 – 20.

Chisman, A. J. (1992). *Introduction to Simulation Modeling Using GPSS/PC, Prentice-Hall International, Englewood Cliffs.*

Cicsplex SM Concepts and Planning (1999), International Business Machines Corporation, New York.

Crain, C. R. (1997). Simulation using GPSS/H, Proceedings of the 1997 Winter Simulation Conference, Uredili: Andradbttir, S., Healy, K. J., Withers, D. H. & Nelson, B. L. Atlanta 7-10 dec. 1997. Piscataway: Institute of Electronics and Electrical Engineers.

Crain, C. R. (1998). Simulation using GPSS/H, Proceedings of the 1998 Winter Simulation Conference, Edited by Medeiros, D. J., Watson, E. F., Carson, J. S., Manivannan, M. S. Washington 13-16 dec. 1998. Piscataway: Institute of Electronics and Electrical Engineers.

Crain, C. R. & Henriksen, J. O. (1999). Simulation using GPSS/H, *Proceedings of the 1999 Winter Simulation Conference, Edited by Farrington, P. A., Nembhard, H. B., Sturrock, D. T. & Evans, G. W. Phoenix 5-8 dec. 1999. Phoenix: Institute of Electronics and Electrical Engineers.*

Gordon G. (1962). A general purpose systems simulator, IBM Systems Journal, **3(1):18-32.**

Guariso, G., Hitz, M. & Werthner, H. (1996). An integrated simulation and optimization modelling environment for decision support, *Decision Support Systems*, **16**(2):103 – 117.

Henriksen, J. O. & Crain, C. R. (2000). GPSS/H: A 23 – year retrospective view, *Proceedings of the 2000 Winter Simulation Conference*, Edited by Joines, J. A., Barton, R. R., Kang, K. & Fishwick, P. A. Orlando 10-13 dec. 2000, Piscataway: Institute of Electronics and Electrical Engineers.

Katwijk, J. & Zalewski, J. (1999). Parallel and Distributed Real-Time Systems: An Introduction, *Parallel and Distributed Computing Practices*, **2**(1):1-6.

Mitzenmacher, M. (1999). On the Analysis of Randomized Load Balancing Schemes, *Theory of Computing Systems*, **32**(3): 361-386.

Murata, T. (1989). Petri nets: Properties, analysis and applications, *Proceedings of the IEEE*, **77**(4): 540– 581.

Rosenthal, J. S. (2000). Parallel computing and Monte Carlo algorithms, *Far East Journal of Theoretical Statistics*, **4**(2): 207-236.

Sang, J., Chung, K. & Rego, V. (1994). A Simulation Testbed based on Lightweight Processes, *Software: Practice and Experience*, **24**(5): 485-505.

Song, B., Ernemann, C. & Yahyapour, R. (2004). Parallel Computer Workload Modeling with Markov Chains, *Proceedings of Job Scheduling Strategies for Parallel Processing: 10th International Workshop*, edited by Feitelson, U. D., Rudolph, L. & Schwiegelshohn, U. New York 13 jun. 2004. Berlin: Springer Verlag.

Wooldridge, M. & Jennings, N. (1995). Intelligent Agents: Theory and Practice, *Knowledge Engineering Review*, 10(2): 115-147.

Žibert, M. (1999). Simulacija delovanja CICS-a na osrednjem računalniku na Zavodu za zdravstveno zavarovanje Slovenije (in Slovenian), Diploma Assignment, University of Maribor, Faculty of Organizational Science.

Žibert, M. (2005). Simulacija delovanja paralelnih enojnih strežnikov v GPSS-u, Master Thesis, University of Maribor, Faculty of Organizational Science.

Marko Žibert received his Master degree in information systems management at the University of Maribor, Faculty of Organizational Science, in 2005. His major research interest includes discrete simulation for the optimization of various information systems. He is employed as a system programmer in the System Department of the Health Insurance Institute of Slovenia (ZZZS).

Miroljub Kljajić is Professor at the Faculty of Organizational Sciences, University of Maribor. His brief biography is published on page 634 of this issue.