Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

STEPS TOWARDS A BETTER INTERNAL GPSS MECHANISM

Ingolf Ståhl

Stockholm School of Economics
Box 6501
S-113 83 Stockholm, Sweden

ABSTRACT

This paper presents the internal procedures used by the micro-GPSS system. They are quite different from those used by standard GPSS, like GPSS/H. Instead of the Future and Current Events Chain, five different lists are used. The advantages of this approach is that queue statistics, both in front of servers and due to WAIT conditions, can be obtained in a far simpler manner, that programs with long waiting lines are executed more efficiently and that, in the case of equal priority, the transaction having waited the longest time in front of a server will always be the first to be served.

1. INTRODUCTION

Recent papers by T. Schriber and D. Brunner (1994 and 1995) have stressed the importance of understanding the internal procedures used by the software of discrete-event simulation for implementing the fundamental management of entities, involving such things as scheduling events and handling waiting lines. In these papers, the approaches taken in this regard by GPSS/H, SIMAN and ProModel are compared, with a focus on the two first languages.

A fundamental difference can be seen comparing GPSS/H and SIMAN. Both systems have their comparative advantages and disadvantages. As regards GPSS/H, a problem is that all events in a waiting stage are handled on **one** single list. This has, in the case of many waiting transactions, the disadvantage of a more cumbersome search procedure. This is only partially hidden by the efficient compilation done by the GPSS/H system. Another disadvantage is that the measuring of waiting line statistics requires separate pairs of blocks, QUEUE-DEPART.

Against this background, this paper wants to bring to attention that another GPSS version, micro-GPSS (**mG**), ever since its first beginning in the 1970s has used a completely different mechanism than the standard approach used by the IBM GPSS versions, by GPSS/PC and by GPSS/H. While these Standard GPSS (**SG**) versions use only two standard lists, or chains, for handling events and waiting lists, called the Current Events Chain (CEC) and the Future Events Chain (FEC), mG uses for these purposes five different types of lists.

2. SOME DIFFERENCES IN SYNTAX

In order to help the novice to micro-GPSS understand the presentation of the five different types of lists, we shall first present some of the fundamental differences in syntax between mG and SG. mG uses only 22 block types, five of which do not exist in SG, namely ARRIVE, GOTO, IF, LET and WAITIF. (See also Ståhl 1996b for more details.)

ARRIVE replaces QUEUE and has (almost) the same syntax; GOTO replaces TRANSFER, but with a simpler syntax; LET replaces ASSIGN and SAVEVALUE (and BLET in GPSS/H); IF replaces TEST and GATE blocks **with** an address, while WAITIF replaces TEST and GATE blocks **without** an address.

In contrast to TEST, IF works with a "straight logic", namely that we proceed to the address in the C operand, if the stated condition is **true** (not false as in SG).

WAITIF works in a similar way: we wait, prior to the WAITIF block, if, and as long as, the stated condition is **true**. The WAITIF condition can concern either a name in the A operand and a server status code in the B operand, or SNAs or constants in the A and B operands. The server status code in the B operand involves the letters EFNU; namely U (in **U**se) or N (**N**ot in **U**se) for facilities; E (**E**mpty), NE (**N**ot **E**mpty), F (**F**ull) and NF (**N**ot **F**ull) for storages. In the SNA case there is an SNA in at least one of the two operands.

3. THE FIVE MICRO-GPSS LISTS

The micro-GPSS processor works with the following five different types of lists:

1. **QLs**, Queue lists, one for **each** server, i.e. facility or storage, handling the waiting lines in front of these servers.

2. **WELs**, i.e. **W**AITIF **E**FNU **L**ists, where EFNU represents the server status codes E, F, NE, NF, NU, U, mentioned above. These lists contain the transactions waiting in front of a WAITIF block with server status code. The correspondence between such WE-lists and WAITIF blocks is discussed below in section 10.

3. One **WSL**, i.e. one **W**AITIF **S**NA **L**ist, where all transactions, waiting in front of a WAITIF block with an SNA in at least one operand, are placed.

4. One **FEL**, i.e. **F**uture **E**vents **L**ist, where all events that are to be executed at a time later than the present clock time are placed.

5. One **REL**, i.e. **R**eady **E**vent **L**ist, where all events that are ready to be executed at the present clock time are placed. All transactions on this list have earlier been on one of the other four lists.

The REL and the FEL are to some extent part of the same list, but we shall for pedagogical reasons first treat them as completely separate lists and in section 11 discuss the actual implementation.

On the QLs, the WELs and the WSL the transactions will as a main rule be sorted primarily on basis of priority, and in the case of equal priority on the basis of time of "true" entry into the list. In certain cases, transactions having just left a list will be put back at the **front** of the list, implying a "dummy" entry. The transactions on the FEL are on the other hand primarily sorted according to event time and only in the case of equal event times on the basis of priority.

On the REL, where all event times refer to the present simulation clock time, all transactions are primarily sorted according to priority. At equal priority, the following applies: The transactions just moved from the FEL are generally put last in each priority group. The remaining transactions are in each priority group sorted according to the entry time into the QL, WEL or WSL from which they have just been removed. Thus, in each priority group on the REL, the transactions brought in from these three types of lists will (almost) always be ahead of the transactions brought in from the FEL (with "entry time" = present clock time; see also section 11). The sorting according to priority is done in descending order, i.e. the transactions with a higher priority number are placed close to the front of the list, while the sorting according to time is done in ascending order, i.e. the lower times are closer to the front.

4. PHASES OF PROGRAM EXECUTION

The **Initialization Phase** is identical to that of GPSS/H (Schriber and Brunner 1994). Here the simulation clock is set to 0 and the initial GENERATE events are placed on the FEL. We here determine as an attribute of the transaction its NB (next block), in this case the block following the actual GENERATE block. This phase finally involves the placement of all events taking place at time 0 on the REL.

In the **Entity Movement Phase** all events on the REL are executed and brought through as many blocks as possible. In the process some events might be brought from a QL, a WEL or the WSL onto the REL. This EMP is described in detail in section 5. As will be discussed there, this phase continues until all events on the REL have been executed and the REL is empty, after which we move to the clock update phase.

The Clock Update Phase: If the FEL is empty, we go to the End Phase. If the FEL is not empty, we do the following: We remove the first item from the FEL and advance the clock to this time. We move this event to the REL, as well as all other events on the FEL that have this clock time as their move time, in such a way that they keep the same order as on the FEL. We then go back to the EMP.

In the **End Phase** all standard reports are printed and control is passed to a superloop in which the simulation program is possibly restarted.

5. THE ENTITY MOVEMENT PHASE

The EMP is subdivided into four steps:

Step 0: We remove the transaction at the front of the REL.

Step 1: We determine the event connected with the NB of this transaction and execute this event. We distinguish between the following types of events:

a **GENERATE**. An IAT is sampled and a new GENERATE event is put on the FEL to happen at the **move time** T= clock time + IAT. The event is put on the FEL, in its priority class, as the last event among the events to happen at time T. We next go to step 2, i.e. to proceed to the next block.

b. **ADVANCE**. The transaction is moved to the FEL with the move time set to the time of exit from the ADVANCE block. It is placed on the FEL according to the same rule as for GENERATE under point a. above. We next go to step 3, since this transaction cannot move further ahead.

c. An **LRR** event (i.e. LEAVE, RELEASE or RETURN) will carry out (i) the QR scheme (Queue Remove scheme) described in section 6, which will move

one transaction, or, in the case of LEAVE, possibly several transactions, from the QL to the REL; (ii) the WER scheme, described in section 8, to see if we shall move transactions from one or several WELs and finally (iii) we go to step 2.

d. An **EPS** event (i.e. ENTER, PREEMPT or SEIZE). We carry out the SE scheme (see section 7) to see if the transaction may enter the block. If it **cannot enter**, we move the transaction into the QL of the server. If the transaction has just come from this QL, i.e. the transaction's JM (Just Moved)-switch is on, we move it to the front of the QL; else it is put as the last member of its priority group on the QL. We next go to step 3. If it **can enter**, we carry out the WER scheme, described in section 8, to see if we shall move transactions from one or several WELs. We finally go to step 2.

e. A **WAITIF EFNU** event, i.e. a WAITIF block in server mode. We here check whether or not the EFNU condition is true. If it is **true**, the transaction is moved to the WEL connected with the actual block (see section 10 below). If the transaction has just come from a WEL, i.e. the JM-switch is on, it goes to the front of the WEL. If the JM-switch is off, the transaction is placed as the last member of its priority group on the WEL. We finally go to step 3. If the EFNU condition is **not true**, i.e. the transaction does not have to wait, we remove the first transaction on the corresponding WEL, if this WEL is not empty, and move it to the REL. We finally proceed to step 2.

f. A **WAITIF SNA** event, i.e. a WAITIF block with an SNA. We here check whether the waiting condition is true. If this condition is **true**, the transaction is moved to the WSL. If the transaction has just come from the WSL, i.e. the JM-switch is on, it goes back to the WSL, to be placed on basis of original entry time into the WSL. If the JM-switch is off, the transaction is placed as the last member of its priority group on the WSL. We finally go to step 3. If the waiting condition is **not true**, we remove the first transaction on the WSL, unless WSL is empty, and move it to the REL. We finally proceed to step 2.

g. An **ASSEMBLE** event. A special process is carried out. We go to step 2 or step 3 depending on whether the transaction can proceed or is stopped.

h. A **SPLIT** event. A number of copies, determined by the A operand, of the transaction are put on the REL, with the NB determined by the B operand. The original transaction goes to step 2.

i. **GOTO** and **IF** without D operand. Since a new NB is determined on the basis of the address in the operands we go directly to step 2b (skipping 2a where a new NB is determined).

j. **TERMINATE**. We decrease the TC (Termination Counter) by the A operand value. If the TC becomes ≤ 0, we go to the End Phase. Otherwise we go to step 3.

k. **PRIORITY**. We change the transaction priority and go to step 3 to put the transaction on the REL, possibly in a new priority class. Another transaction on the REL might now have a higher priority.

l. All **other** events will be executed without having any effect on lists or the Next Block. We here proceed to step 2.

Step 2: The transaction can proceed forwards to another block at this clock time. We have three substeps: 2a. We increase NB by 1; 2b. We carry out the WS scheme (WAITIF SNA scheme) described in section 9, thereby possibly moving transactions from the WSL to the REL and 2c. We go back to step 1.

Step 3: We carry out the WS scheme. If REL is empty, we go to the CUP (Clock Update Phase). If REL is not empty, we remove the first event from the REL, establish its NB and go to step 1 with this event.

6. THE QR SCHEME

The QR (**Q**ueue **R**emove) scheme moves one or several transactions from the QL to the REL. The processor tests if there are any events on the QL associated with the server referred to in the LRR block. If the QL is not empty, we distinguish between two cases of action:

a. The case of either a RELEASE-RETURN block or of a LEAVE block when there is not any B>1 in the program, implying that the transaction freeing the server allows room for exactly **one** other transaction. In this case we remove the first transaction on the QL and move it to the REL, placing it, in its priority group, on the basis of the time of entry into the QL.

b. The case of LEAVE in a program where there is some B>1. We here have two subcases:

(i) The LEAVE block has B=1. We start the search in the QL from the front for the first transaction that has an ENTER with B=1. This transaction is then removed from the QL and moved to the REL, in the same way as under 6a.

(ii) The LEAVE block has a B>1. We start a search in the QL, from the front, determining for each transaction on the list, which all involve ENTER events, a value B_e of the B operand of this ENTER block. If $B=B_e$, the same action takes place as under (i) and we are finished with the QR scheme. If $B<B_e$, we continue to the next transaction. If $B>B_e$, we decrease B by B_e and move the transaction from the QL to the REL, but continue to the next transaction with this procedure, until we reach the last transaction on the QL or until B=0.

7. THE SE SCHEME

The SE (Server **E**ntry) scheme deals with an EPS block. In the case of SEIZE, the transaction may enter if the facility is idle. In the case of ENTER, the transaction may enter if the free capacity of the server $\geq$ B. In the case of PREEMPT, the transaction may enter if the facility is idle. It may also enter if the facility is busy by a SEIZE block, or, in the case of PREEMPT A,PR, by a transaction with a lower priority. In the latter two cases we carry out a special pre-emption activity. It is outside the scope of this paper to discuss this activity.

In all of these three cases, if a transaction can enter the server, it proceeds to the next block. If it cannot enter, its behavior depends on if it has come to the REL from the FEL or from a QL. In the FEL case the transaction is put on the QL as the last member of its priority group. In the QL case it is put back at the front on the QL, i.e. at the position it just held.

8. THE WER SCHEME

The WER (**WE** list **R**emove) scheme will possibly move a transaction from one or several WELs to the REL. A certain server is connected with each LLR or EPS block. With each server there is, in turn, a connection to one or several WELs (this association is discussed further in section 10 below). Having established the server of the block, we next look through **all** WELs associated with this server. For each such WEL there is in turn associated a specific WAITIF condition. If this WAITIF condition is fulfilled, no action will be taken (i.e. waiting will continue). If the WAITIF condition is **no longer** true, we will remove the first transaction of the WEL and move it to the REL, where it is placed on the basis of the start wait time in this WEL.

9. THE WS SCHEME

The WS (**W**AITIF SNA) scheme is carried out, if there is at least one WAITIF SNA block in the program. We first check whether or not the WSL is empty. If the WSL is empty, we proceed to the next task. If the WSL is not empty, we start from the front and go backwards through the whole list. For each transaction we check whether or not the wait condition still holds. If it still holds we proceed to the next transaction. If the wait condition does **not** hold, we remove this transaction from the WSL and put it in its priority group on the REL, placing it here on the basis of time of entry into the WSL. In this way we continue until all transactions on the WSL have been examined.

10. WELS AND WAITIF EFNU BLOCKS

When discussing the relationship between WELs and WAITIF EFNU blocks, we shall distinguish between the following three cases as regards the A operand of the WAITIF block:

a. The A operand is a name. All blocks that have the same server name **and** the same one of the six status codes (E, F, NE, NF, NU or U) share the same WEL. Transactions having to wait at this WEL will, if they have the same priority, be sorted according to when they started to wait due to this specific condition.

b. The A operand is a parameter referring to a **facility**. We allow in this case both for the case of P*j*=U and P*j*=NU, for a certain number, *n*, of integer values of the parameter P*j*, thus allowing for *n* WELs of 1=U, .., *n*=U, and for *n* WELs of 1=NU,.., *n*=NU. Every block WAITIF P*j*=U will then refer to the first mentioned *n* lists and every block WAITIF P*j*=NU will refer to the second mentioned *n* lists.

c. The A operand is a parameter referring to a **storage**. Since this kind of usage is rare, we allow in this case only for **one** single WEL. The first WAITIF condition referring to E, F, NF or NE in the program will use a number of WELs and every WAITIF block with a parameter in the A operand and the same status code in the B operand will refer to these WELs. If there then, later in the program, is another WAITIF with a parameter in the A operand, but another status code in the B operand, then this WAITIF EFNU block will internally be treated as a WAITIF SNA block and the transactions waiting at this block will be waiting on the WSL.

This can be exemplified as follows: Assume we first have a block WAITIF P1=E and later a block WAITIF P3=F. Then the transactions waiting before the first block will do that on a number of WELs, for 1=E, 2=E, etc. The transactions waiting in front of the other block will be waiting on the WSL as long as R(P3) =0, i.e. as long as the remaining free capacity of the storage referred to by P3 is 0.

11. IMPLEMENTATION OF REL AND FEL

The REL and FEL together constitute an EL, Events List, with all transactions on the REL at the front of the EL and all transactions on the FEL at the end of the EL. Each transaction on this EL has a pointer to the preceding member and a pointer to the succeeding member, except that the first member has no predecessor and the last member no successor. We have one list pointer to the first member and one to the last member on this EL. We also have a special R[illegible]L pointer, the

RELL (REL **L**ast) pointer, pointing at the last member of the REL. The successor of this member is thus the first member of the FEL. The movement of all events taking place at the new simulation clock value in the CUP can now be done in a very simple fashion. We just have to establish, starting with the successor to the old RELL pointer, the last new member on the REL and then change the RELL pointer to point at this transaction.

When we bring in a new transaction from a QL, a WEL or the WSL to the REL, we will start to move it from the back of the REL, i.e. from the transaction at which the RELL pointer points. We will put it in front of any transaction with a lower priority and, in the case of equal priority, with a lower waiting time, i.e. with a higher entry time into the original QL, WEL or WSL. Since the transactions which in the CUP were brought in from the FEL all have 0 waiting times, the new transaction will, in the case of equal priority, be put in front of these FEL transactions, provided it has a waiting time >0.

An advantage with the joint approach is that all removals from this EL will take place from the front and can be done by the same procedure. When we remove the first transaction from the REL, the front of REL is of course the front of the EL. When we remove the first transaction from the FEL, the REL is empty and we hence remove the front transaction from the EL.

12. COMPARING WITH THE SG APPROACH

In Ståhl (1996a) we show that, except for four specific differences, to be mentioned below, the procedure presented above for mG will result in the same execution results as that of SG. We there compare the mG approach with the SG approach as presented by Schriber, in 1974 for GPSS/360 and in 1990 for GPSS/H.

We shall here only mention the following: The Initialization Phases are equivalent, with the only difference being that the events put on the FEC in SG are put on the FEL in mG. Likewise the CUPs are very similar. In both cases, we update the clock to the event time of the first transaction on the FEL/FEC and move **all** events happening at this updated clock time from the FEL/FEC to the REL/ CEC, before proceeding to the EMP. One, in reality unimportant difference, to be called **difference 1**, is that execution stops with a normal report in mG, when the FEL is empty, while it stops with an error report in SG. The End Phase is also similar, in that all standard reports are printed, but the actual reports vary in format.

All the obvious differences happen in the EMP. One difference between mG and most SG versions (GPSS/H and GPSS/V, but not GPSS/PC) deals with when the arrival of, for example, customer 2 at a GENERATE block is put on the FEC/FEL. In SG this is done when customer 1 is moved **out of** the GENERATE block, but in mG this is done when customer 1 moves **into** the GENERATE block. This difference, **difference 2**, will matter greatly if the GENERATE block is followed immediately by, for example, SEIZE. In GPSS/H, the introduction of an ADVANCE block with no operands between GENERATE and SEIZE would then make a big difference. In mG such an ADVANCE block would in this case not matter.

Such an ADVANCE block without any operand, will in other cases lead to another difference. Let us look at the following simple program in SG (GPSS/H). (For the corresponding mG code, see Ståhl 1996a).

```
         SIMULATE
NUM      GENERATE    20
         ASSIGN      1,N$NUM
BEGIN    ADVANCE
         SEIZE       RIDE
         ADVANCE     20
         RELEASE     RIDE
         TRANSFER    .5,,BEGIN
         TERMINATE
         GENERATE    101
         TERMINATE   1
         START       1
         END
```

Program example 1

The result of this program will be different in SG and mG. The first customer will in SG repeat his ride immediately, while in mG the second customer will immediately get a ride. The reason for this is that in mG, entry into the (BEGIN) ADVANCE block will always cause the transaction to be placed on the FEL and the next transaction to be taken from the REL or FEL. ADVANCE without an A operand hence does the same thing as BUFFER or YIELD in GPSS/H.

This is **difference 3**. It should be noted that if we add an A operand, e.g. 0.00001, to the ADVANCE block the results will be identical in mG and SG, with the second customer immediately getting a ride. Likewise, if we replace (BEGIN) ADVANCE with, for example, a simple assignment (ASSIGN VAL,0,XL in SG; LET X$VAL=0 in mG), SG and mG will also yield identical results, now with the just released first customer getting his second ride right away.

13. FIFO IN WAITING LINES IN MG, BUT FIFO ON THE CEC IN SG

In mG, the transactions blocked by an EPS-block referring to a specific server are, as noted, all put on a specific QL, where those transactions that have the same priority are sorted according to their time of entry into this QL. For the simple case of a facility, where all entries are made with SEIZE, and of storages, where all entries are made with ENTER with B=1, we will then always remove the first transaction from the QL. This implies, in the case of equal priority for all transactions, a straight FIFO-discipline as regards the waiting line in front of a server. A transaction will in this case never go ahead of another transaction that has waited a longer time in front of this **specific server**.

This FIFO discipline with regard to the server might, however, be violated in SG. This constitutes **difference 4**. This is due to the fact that the transactions on the CEC are never resorted while they remain on the CEC, except for the above mentioned case with PRIORITY. The only other way they can be resorted is by moving them to the FEC, e.g. to be have an exit from an ADVANCE A block scheduled, and then return at the exit from this block. Once they return from the FEC, they are put at the end of their respective priority group.

Let us as an example assume that a transaction, customer 1, at time 20, is first blocked trying to seize one facility, FACA, and upon being unblocked from this facility at time 25, **immediately** tries to seize another facility, FACB, which at this time is busy. Another transaction, customer 2, will at time 23, immediately when generated, try to seize facility FACB, which is busy already at this time.

In SG, customer 1 remains waiting for facility B on the position determined by having come to the CEC at time 20. Customer 2, who is transferred from FEC at time 23 to seize FACB B directly, without having to first use facility A, is then placed on the CEC **behind** customer 2. Since facility B was busy also at time 23, customer 2 starts waiting for facility B **earlier** than customer 1. However, when facility B becomes free, e.g. at time 30, the rescan starts from the beginning of the CEC and hence customer 1 gets the chance to move first to seize facility B that has now become idle, in spite of the fact that customer 2 has been waiting in front of facility B for a longer time.

In mG, customer 2 will be placed on the QL of FACB at time 23 and customer 1 on this QL at time 25. Customer 1 is hence placed behind customer 2 on this QL. When FACB becomes free at time 30, customer 2, who has waited the longest will be served.

The mG approach appears as more intuitively appealing, since it implies that the person having waited the longest time among those with equal priority in front of a specific server will always be served first. That some servers are needed jointly under some, but not all, circumstances does not immediately imply that there is a joint waiting line. It also appears easier to learn that each server has its own waiting line, with FIFO-discipline in case of equal priority. To understand the SG approach one must really understand the CEC approach.

It is furthermore easier to understand the idea of how to get a joint waiting line in mG than of how to get separate waiting lines in SG. To get a joint waiting line in mG in the example above, one just puts a block PRIORITY A, with A>0, implying a **higher** priority, between SEIZE FACA and SEIZE FACB. To get separate waiting lines in front of FACA and FACB in GPSS/H, one could also include a block PRIORITY A between SEIZE FACA and SEIZE FACB, but with A=0, i.e. implying **unchanged** priority. Why this is so also requires good understanding of the CEC mechanism.

The problem that waiting in SG is bound to the CEC and not the waiting lines is perhaps even more clear in program example 2, presented below in its GPSS/H form.

```
        SIMULATE
EXHIB   STORAGE     5
        GENERATE    10
        TEST GE     C1,100
        ASSIGN      1,C1
        ENTER       EXHIB
        ADVANCE     120
        LEAVE       EXHIB
        TERMINATE
        GENERATE    10
        ASSIGN      2,C1
        ENTER       EXHIB
        ADVANCE     120
        LEAVE       EXHIB
        TERMINATE
        GENERATE    400
        TERMINATE   1
        START       1
        END
```

Program example 2

We here have two kinds of visitors coming to an exhibition; invited guests and ordinary visitors. The ordinary visitors are allowed into exhibition building only after 100 minutes. We assume that all visitors stay for two hours. There can only be five visitors at a time in the small exhibition room, talking to the artist. Visitors of each type arrive 10 minutes apart. When a visitor enters the exhibition room, we save, in P1 for ordinary visitors and P2 for invited guests, the time that he started to wait at the entrance of the exhibition room.

In the SG case, an ordinary visitor who started to wait at the exhibition room at time 100 is allowed to go into the room before an invited guest who arrived and started waiting at time 60. In the corresponding program in mG this will not happen, but the people waiting at the exhibition room enter this in strict first come-first served order.

14. OTHER ADVANTAGES OF THE MG APPROACH

There are two other substantial advantages of the mG approach besides the strict adherence to the FIFO-principle dealt with in section 13. The first and most important one is the easy way in which gathering of queue statistics can be coded.

The gathering of queue statistics referring to a facility SERV would in SG require three blocks: QUEUE SERV, SEIZE SERV and DEPART SERV. In mG it is enough to write SEIZE SERV,Q. The same Q as B operand (or possibly C operand) can also be used in connection with ENTER.

We can now in micro-GPSS obtain queue statistics in a simplified fashion also in connection with a WAIT condition by giving the name of the queue as a C operand of WAITIF. Thus, the gathering of statistics in front of a GATE, which in SG would require e.g. the three blocks QUEUE QSTUD, GATE SNE STOCK and DEPART QSTUD, can in mG be handled by one single block WAITIF STOCK=E,QSTUD.

The saving in number of blocks is substantial. As an example, we have rewritten all programs in Schriber's famous "red book" from 1974 using mG. In many cases we have with the 22 blocks of mG been forced to use several blocks to replace a non-existing SG block, e.g. LOOP. Yet for the 29 programs presented in the book, the average number of blocks used is virtually the same (18.6 in SG and 18.8 in mG). This is mainly due to the fact that we have saved on average 1.24 blocks per program thanks to this simplified queue statistics gathering, based on separate QLs.

The other advantage is that the program execution of mG is in principle more efficient than that of SG. This is quite clear when comparing mG with SG systems such as GPSS/PC. It is less easily evident when comparing with GPSS/H, which is a compiled system and focused on rapid execution. Yet there are examples when mG, although interpretative and not so well coded for optimization, will execute faster than GPSS/H. This refers to cases with very long waiting lines, like in program example 3, presented here in GPSS/H.

```
        SIMULATE
        INITIAL     XL$MAX,2000
        GENERATE    1.8,0.6
        TEST NE     Q$SAL,XL$MAX,BYE
        QUEUE       SAL
        SEIZE       SAL
        DEPART      SAL
        ADVANCE     25,5
        RELEASE     SAL
BYE     TERMINATE
        GENERATE    1,0.6,,,1
        QUEUE       SUL
        TEST NE     Q$SUL,4,BYTT
        PREEMPT     SAL
        DEPART      SUL
        ADVANCE     1,0.5
        RETURN      SAL
        TERMINATE
BYTT    DEPART      SUL
        TERMINATE
        GENERATE    32000
        TERMINATE   1
        START       1
        END
```

Program example 3

Although GPSS/H is very efficient when skipping all transactions for which the Scan Indicator is on, the computer still has to investigate, in the EMP, **every** single transaction on the CEC, e.g. at RELEASE, to check the status of the Scan Indicator. In mG it is enough to remove the very first transaction from the QL of SAL. When used by a compiling system, focused on optimization, the separate QL approach should be faster also in the case of much shorter waiting lines. For better speed, one would in GPSS/H in this case have to use a more complicated and less-easy-to-learn LINK/UNLINK construction, allowing for a non-standard road to the more efficient approach with separate QLs.

Against this background we shall finally argue that the approach with separate QLs for each server, and separate WELs for most blocks of the WAIT type, should be made standard in future languages of the GPSS-type.

ACKNOWLEDGEMENTS

The ideas behind this paper, as well as the development of micro-GPSS, have been greatly influenced by Tom Schriber, not only through his publications, listed in the references, but also through his comments, by voice, mail and Email.

REFERENCES

Schriber, T. J. 1974. *Simulation Using GPSS*, Wiley, N.Y.

Schriber, T. J. 1990. *An Introduction to Simulation with GPSS Using GPSS/H*, Wiley, N.Y.

Schriber, T. J. 1995. How Discrete-Event Simulation Software Works. In *Eurosim'95 Simulation Congress*, ed. F. Breitenecker and I. Husinsky, 17-28. Elsevier, Amsterdam.

Schriber, T. J. and D. T. Brunner. 1994. Inside Simulation Software: How It works and Why It Matters.In *Proceedings of the 1994 Winter Simulation Conference*, ed. J.Tew, S. Manivanna, D. Sadowski, and A. Seila, 45-54. SCS, La Jolla.

Schriber, T. J. and D. T. Brunner. 1995. Inside Simulation Software: How It works and Why It Matters. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopoulos, K.Kang, W. Lilegdon and D. Goldsman, 451-456. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Ståhl, I. 1990. *Introduction to Simulation with GPSS: On the PC, Macintosh and VAX*, Prentice Hall International, Hemel Hempstead, U.K., 1990.

Ståhl, I. 1995. *Simulation Made Simple with micro-GPSS: A Short Tutorial with Seven Lessons*, Stockholm School of Economics, Stockholm.

Ståhl, I. 1996a. *Steps Towards a Better Internal GPSS Mechanism*, EFI Working Paper, Stockholm School of Economics, Stockholm.

Ståhl, I. 1996b. Teaching the Fundamentals of Simulation in a Very Short Time. In this volume.

AUTHOR BIOGRAPHY

INGOLF STÅHL is Professor at the Stockholm School of Economics, Stockholm, and has a chair in Computer Based Applications of Economic Theory. He was visiting Professor, Hofstra University, N.Y., 1983-1985 and leader of research project on inter-active simulation at the International Institute for Applied Systems Analysis, Vienna, 1979-1982. He has taught GPSS for twenty years at universities and colleges in Sweden and the USA. He has on the basis of this experience led the development of the micro-GPSS system. He is also consultant in simulation to Swedish banks and industry.