Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

GPSS*
A GPSS IMPLEMENTATION WITH HIERARCHICAL MODELING FEATURES

Thomas Behlau
Volkmar Hinz

Technical University Magdeburg
Department of Computer Simulation and Graphics
PSF 4120
D - 39016 Magdeburg, Germany

ABSTRACT

Though GPSS is about 32 years old it has not been changed in its main algorithms since Henriksens new version (see Henriksen 1976). This paper describes some new features of a GPSS implementation in order to improve the macro concept of GPSS. Besides a short review on environments for simulation systems is given. Finally an OS/2 environmental frame for the new GPSS* is suggested.

1 INTRODUCTION

At present GPSS belongs to the most used discrete simulation systems in the world. However, its internal structures have not been modified since the developments by Henriksen in 1976. But the current fad in computer science and simulation breaks with the classical simulations languages and goes in the direction of graphical oriented simulation environments. Beside the lack of any environment GPSS has a number of disadvantages presented in the following paragraph:

GPSS does not allow hierarchical modeling. GPSS provides structured programming with macros using parameters to specify the entities in the macro. However, all names are used in a global mode. That is why a real hierachical modeling is impossible.

GPSS does not have an array concept. Only a specialized EQU statement can be used
(e.g. DESK EQU 5(3),F,Q).

GPSS does not allow the addition either of new entity classes (transaction, queue, storage) or of new methods (ADVANCE, QUEUE, ENTER). Enlargements are possible with the help of subroutines in FORTRAN or C only.

Because *the user can not see the model entities* it is more difficult to teach and to learn GPSS than other simulation languages. The user only works with the entities using statements to manipulate them.

Transactions cannot be provided with names. For that reason debugging of larger programs is difficult. Transactions can be distinguished either by different priorities or specialized parameter values.

GPSS programms can hardly be structured clearly. This criticism is well–founded considering that parameters cannot be transfered to subroutines by using TRANSFER SBR as in other programming languages. The fact that in GPSS several transactions can be processed at the same time must be kept in mind and therefore only their parameters are candidates for parameters of subroutines.

This paper adresses the deficiencies of the macro concept as well as the lack of a modern environment for the development of simulation projects.

2 DISADVANTAGES OF THE MACRO CONCEPT OF GPSS

To reduce errors writing the source code of a simulation program GPSS provides macros. A macro definition is a sequence of GPSS statements where portions of the statements can be replaced by operands. These operands are supplied each time the macro is invoked. (see GPSS/H manual).

However, the concept has some deficiencies:

1. The macro definition has to be located in the simulation source code. There is no way to construct libraries of sets of macros.

2. Each time a MACRO definition statement is found the compiler produces a sequence of GPSS statements where the text to be replaced — coded by a # character followed by a letter (A–J) — is substituted with the corresponding macro definition operand. But this procedure is nothing more than a string exchange. In reality no simulation code is saved up.

3. This string replacement shows disadvantages if

one uses macros which contain other macro directives. A simple example should illustrate that.

```
MACHINE STARTMACRO
        QUEUE       #A
        SEIZE       #A
        DEPART      #A
        ADVANCE     #A
        RELEASE     #A
        ENDMACRO
*
MCENTRE STARTMACRO
        QUEUE       #A
MACHINE MACRO       MACH1,30,10
        ADVANCE     #B,#C
MACHINE MACRO       MACH2,20,15
        ADVANCE     #E,#F
        DEPART      #A
        ENDMACRO
```

Now two macros are invoked by means of the MACRO directives

```
MCENTRE MACRO  MCENTRE1,20,10,30,20
MCENTRE MACRO  MCENTRE2,15,7,20,15
```

There will be an undetected error using facilities and queue with equal names both in the macro MCENTRE1 and macro MCENTRE2. With a more sophisticated programming one can avoid these errors but they will become more likely using higher levels of nesting.

However these problems are not only typically for GPSS, but also for other simulation languages as SIMAN, SLAM etc.

3 HIERARCHICAL MODELING APPROACHES

There are a lot of extensions to existing simulation systems which try to overcome the disadvantages of the macro concepts. Granas created a GPSS-like hierarchical simulations language in 1986. But this system was not able to work with older GPSS programs. It was a real new simulation language.

Pedgen and Davis introduced the system Arena^{TM}, a hierarchical system, based on the SIMAN simulation language. The use of hierarchical features in this system are restricted to the environment, creating a SIMAN source program. The language itself is not extended.

In general, macro concepts generate block statement every time a macro is invoked. This is very inefficient, because blocks only need to be coded once per macro.

The main difference between these two approaches above and the new concept of GPSS* are

- full compatibility with the old GPSS language
- inserting new features in GPSS as new statements
- use of local entity.
  This is very similar to the declaration of variables in programming languages (i.e. PASCAL). Identifiers of entities are only valid in the block where they are defined.
- coding block statements only once for all instances of a submodel. Only data of submodels are generated more than once.

4 HIERARCHICAL MODELING USING SUBMODEL'S

The new features — named *SUBMODELS*— should overcome the following deficiencies:

1. define new entities
2. define new blocks and control statements working with these new entities
3. define new standard numerical and logical attributes (SNA's and SLA's)
4. use local entities
5. reduce errors of name conflicts
6. build libraries of new entities

In the next subsections statements corresponding with the submodel entity are described.

4.1 Definition of Submodels

The statement sequence

```
SUBMODEL name
    PREFIX ll
    PREAMBLE
    ENDPREAMBLE
    ********************************
    model the system behavior
    ********************************
ENDSUBMODEL
```

is used to define a submodel *name*. The *PREFIX* statement can be used to substitute the name of the submodel with a short name which is used with standard attributes.

4.2 Declaration in the PREAMBLE

With the *PREAMBLE* statement a sequence of statements starts which defines new model entities, names of new blocks, control statements and standard attributes. The entire form of this statement can look as follows:

```
PREAMBLE
    LOCAL
    ***** define local entities
    ENDLOCAL
    GLOBAL
    ***** define global entities
    ENDGLOBAL
    LABELS
    ***** define new block names
    ENDLABELS
    CONTROLS
    ***** define new control statement
    ***** names
    ENDCONTROLS
    ATTRIBUTES
    ***** define new SNA's and SLA's
    ENDATTRIBUTES
ENDPREAMBLE
```

The parts *LOCAL* and *GLOBAL* are used to define local und global entities of the simulation model which is comparable with subroutines in Pascal. All names of entities which are declared between the *LOCAL* and *ENDLOCAL* statement are only valid in this submodel and local submodels. Names of entities which are defined *GLOBAL* has to be already declared in the main simulation model or in a submodel of a higher level. The references to these entities will be established during compilation. New names for blocks, control statements and standardattributes are declared in the parts *LABELS, CONTROLS* and *ATTRIBUTES*. The exact syntax for these statements is

```
LABLES
    labelname param1,param2,...
ENDLABELS
CONTROLS
    controlname param1,param2,...
ENDCONTROLS
```

and for attributes:

```
ATTRIBUTES
    attrname ll
ENDATTRIBUTES
```

A list of formal parameters has to be given with the block and control statement description. Each formal parameter should be defined as a local model entity.

4.3 Definition of the System's Behavior

The definition of the system's behavior includes the following tasks

- description of the SNA's and SLA's
- programming new *LABELS* and *CONTROLS*
- description of actions for *CLEAR* and *RESET*
- description of a standard output
- definition of other structurs like
  - describe the flow of internal transaction
  - define algorithms for scheduling and control

The next pattern can be used to define a block with a specified name.

```
labelname   blockstatement   operands
            blockstatement   operands
            ...
            blockstatement   operands
            GOOUT
```

The new block statement *GOOUT* cause the transaction to return to the main simulation model or to the submodel calling this submodel.

A similar way is used for control statements. The *CLEAR* and *RESET* statements have to be defined.

The description of new standard numerical and logical attributes can be compared with the declaration of a variable in GPSS/H. After the name of the attribute an arithmetical or locical expressionis following which is evaluated with every call of the SNA or SLA.

The desription of the system's behavior is ending with the definition of the standard output. The standard output is declared in a *REPORT* statement, with the following format:

```
REPORT
    HEAD
    **** output of a title for all
```

```
        **** submodels of this type
        ENDHEAD
        ENTITY
        **** output for every entity of
        **** this submodel type
        ENDENTITY
    ENDREPORT
```

With all these new statements you can create new entities put them into libraries and use them in other models. There are no differences between new block statements and the basic statements of GPSS.

5 AN EXAMPLE OF A SUBMODEL

This examble is the same as in section 2.

```
*
*  Creating a new entity MACHINE
*  with the blocks INMACH
*
   SUBMODEL MACHINE
   PREFIX MA
     PREAMBLE
       LOCAL
         MACH  : FACILITY,QUEUE
         &MEAN,&STD  : REAL
       ENDLOCAL
       LABELS
         INMACH  &MEAN,&STD
       ENDLABELS
     ENDPREAMBLE
*
*    Definition of the block
*
  INMACH  ASSIGN  1,&MEAN-&STD+2*FRN1*&STD,PL1
          QUEUE   MACH   go into the local queue
          SEIZE   MACH   seize local facility
          DEPART  MACH   free queue
          ADVANCE PL1
          RELEASE MACH   free facility
       GOOUT
*
*   Definition of the control statements
*
      CLEAR   CLEAR   ALL
      RESET   RESET   ALL
ENDSUBMODEL    MACH
*
*   Definition of the new entity MCENTRE
*   with the block INMCENTRE
*
   SUBMODEL MCENTRE
     PREFIX MC
       PREAMBLE
         LOCAL
           MCENTRE : QUEUE
           MACH1,MACH2  : MASCHINE
           &MEAN1,&STD1,
           &MEAN2,&STD2  : REAL;
         ENDLOCAL
       LABELS
         INWERK   &MEAN1,&STD1,&MEAN2,&STD2
       ENDLABELS
     ENDPREAMBLE
*
*   Defintion of the block INMCENTRE
*
INMCENTRE ASSIGN  1,&MEAN1-&STD1+2*FRN1*&STD1,PL
          ASSIGN  2,&MEAN2-&STD2+2*FRN1*&STD2,PL
          QUEUE   WERK
          INMACH  MACH1,30,10
          ADVANCE PL1
          INMACH  MACH2,45,25
          ADVANCE PL2
        GOOUT
*
*   Defintion of Control statements
*
      CLEAR CLEAR ALL
      RESET RESET ALL
ENDSUBMODEL
```

The statements

```
INMCENTRE   MCENTRE1,20,10,30,20
INMCENTRE   MCENTRE2,15,7,20,15
```

will not detect errors as in the macro statements because the specific entites are defined to be local. The notation of the source is more similar to the standard source code of a GPSS programm.

These new structures are implemented for OS/2 using the System Object Model (SOM) features. This new GPSS — called GPSS* — requires special needs to an environment. The next section describes a prototype of a powerfull simulation environment.

6 A SIMULATION ENVIRONMENT FOR GPSS*

Using GPSS simulation systems for applications most users demand

- a better support in the model development (development environment) and
- powerful tools for preprocessing external model input parameters or to collect and prepare simulation results which are easy to use (high-level I/O-interface).

Available GPSS implementations (i.e. GPSS/H^{TM} (Wolverine), GPSS/PC^{TM} (Minuteman)) are extremely bad concerning these requirements.

Since years development environments are fully integrated in high-level programming languages (Borland's "IDE", Microsoft's "PWB"). However, GPSS implementations were put on the market without any environment (i.e. GPSS/H) comparable or without a modern acceptable environment (i.e. GPSS/PC), respectively.

Today the GPSS user must create a model as follows:

- edit with a general purpose text editor
- run GPSS Compiler
- read compiler listing and note the error messages
- edit the source and so on.

In available GPSS implementations I/O functionality is on a low level. In GPSS/H the I/O is organized with the help of GETLIST and PUTPIC statements (only sequential access) or by using external routines (HELP, CALL, BCALL). In GPSS/PC one can use the HELP block statement only (calls a external routine written in FORTRAN or PASCAL).

There are no tools as for instance for the processing dBase files or to prepare simulation results for statistical packages existing.

In addition to the implementation of GPSS* described above a simulation environment has to be developed.

This is for the reason of better support in the development of GPSS* models. Besides, preprocessing input parameters as well as preparing output results are possible. This idea is not limitted to GPSS* but partly applicable to other GPSS implementations.

The GPSS* simulation environment should be divided in the parts

- GPSS* model development environment (MDE)
- GPSS* submodel package for high-level I/O

6.1 GPSS* developmental environment

GPSS* is a simulation system familar with GPSS. It is running under the OS/2 2.x environment (32 bit application). GPSS* for OS/2 consists of

- GPSS* kernel (dynamic link library)
- GPSS* metacode generator (processed GPSS-V code with many extensions)
- GPSS* run time processor (processing the metacode with the GPSS* kernel)

GPSS* models are implemented in GPSS source code. The model implementation by composing GPSS block symbols with the help of a graphical environment is not an acceptable way, especially for the development of large and complex models. However, those editors are suitable for teaching the GPSS modelling process. The basis of the development environment for GPSS* should be based on a powerful text editor. Analysing development environments of high-level programming languages we propose a text editor extended by function groups below:

Function group 1: (editor functions)

- formatting of GPSS source code
- syntax expansion (optional)
- keyword help
- syntax scan of source lines (optional)
- syntax highlighting

Function group 2: (environment functions)

- invoke the GPSS* metacode generator (compiling)
- start GPSS* simulation
- view compiler listing
- view and prepare standard output
- maintenance of GPSS* SUBMODELs

Together with the OS/2 2.x operating system there is a powerful text editor called "enhanced editor" (EPM) supplied. The "enhanced editor" itself is an application of the E-Toolkit built by two dynamic link libraries (ETKExxx.DLL, ETKRxxx.DLL). The E-Toolkit supports the software developer by building applications that edit multiple lines of text (see E-Toolkit manual). For building editor based applications two fundamental ways exist. At first one can write a new Presentation Manager (PM) application like EPM with own extensions. Secondly the existing application EPM can be extended by writing editor macros using the editor language "E". The prototype of the GPSS* Simulation Environment was implemented by using the second way. Through EPM macro GPSS.EX the following extensions for EPM are available:

Extensions for fuctions group 1:
Linking GPSS.EX into EPM adds a new menu item "GPSS* Editor Settings..." to the action bar pulldown "Options". It displayed a Notebook to change any editor settings especially for GPSS*.

For GPSS* source code files (*.GPS or *.GSS) EPM used the tabulator positions (2,8,19,40,50,60). The values for comment positions (40,...) are changeable in the "GPSS Editor Settings..." Notebook.

While writing a source line a syntax expansion occurs when Space or Enter is pressed. The kind of syntax expansion depends on the selected Syntax Expansion Level (Settings Notebook) listed below:

Level 0 : no syntax expansion
Level 1 : expansion of block and control statements, jump to the next tab position
Level 2 : operand field generation in addition to level 1

By pressing CTRL-H a help for any GPSS* keyword is available. The option "Scan" (Settings Notebook) specifies that each line is to be checked for proper syntax as it is entered. (Set option to "ON").

Extensions for Function group 2:
Linking GPSS.EX into EPM causes a new action bar entry"GPSS* Simulations..." with menu items

"Compile model",

"Start simulation"

"Prepare output",

"GPSS* system settings..." and

"Submodel maintenance".

The item "Prepare output" starts the GPSS* output browser to view and prepare GPSS* standard output. The item "GPSS* system settings..." displays a notebook for changing any GPSS* compile time and run time settings (i.e. protocol form, actions by errors, automatically start output browser).

"Submodel maintenance" consists of "List exports" and "Library maintenance". "List exports" opens a dialog with informations about a SUBMODEL of a given name. Informations are including

- exported Block statements and SNA's/SLA's
- operands of exported Block statements
- general informations of submodel (date, time, version)
- user comment

"Library maintenance..." supports GPSS* submodel libraries (.GSL, contains precompiled submodels) by

- add, replace, delete of precompiled submodels (.GSO) and
- edit submodel sources (.GSS).

6.2 GPSS* submodel package for high-level I/O

The GPSS* submodel package is a submodel library. An high-level I/O submodel exports a set of block statements and SNA's/SLA's. The implementation of high-level I/O is based on combination of GPSS* code and dynamic linking of external procedures with the HELPDLL block statement. A first prototype submodel for the access to dBASE-files (dBASE-III or IV) exports block statements

DBUSE	open dBASE-file as a database
DBCLOSE	close a database
DBREAD	read the actual record
DBWRITE	write the actual record
DBSEEK	seeks to a record number
DBLOCATE [x]	locate to a matched record

and standard numerical/logical attributes

DBFXn	number of fields of each record
DBRXn	number of records of database
DBFTn(i)	type of field i
DBFVn(i)	value of field i (if numerical)
DBRNn	actula record number
DBEOFn	TRUE if end of file

The SNA's DBF?n(i) contents the values of the actual record.

Analog it is planed to implement a set of submodels for preprocessing many kinds of input data or for preparing output data for statistical packages existing.

7 DISCUSSION

This paper describes some new features of a modern GPSS. With this new extensions the user of GPSS* can add entities to GPSS as well as new block and control statements. The new GPSS* is implemeted using the OS2 system. That is why a new simulation environment is introduced for this platform. An editor was built with the E Editor Toolkit of IBM. This editor allows both text manupulation and submodel maintenance.

With the new GPSS* and its environment a simulation product was developed combining the old fashioned GPSS with techniques of modern programming languages.

REFERENCES

M. Granas, F.J. Torrealdea, A. D'Anjou, and F. Ferretes. *Semantic description of a GPSS-like hierarchical simulation language.* Proceedings of the 2nd European Simulation Conference, 1986.

J. O. Henriksen.*Building a better GPSS. A 3:1 performance enhancement.* Proceedings of the 1975 Winter Simulation Conference.

J. O. Henriksen.*The Development of GPSS/85.* Proceedings of the 2nd European Simulation Symposium, 1985.

Henriksen,J. O. and R. C. Crain. 1989 *GPSS/H Referenz Manual*, Third Edition. Wolverine Software Corporation, Annandale, VA.

IBM T.J.Watson Research Center. *Programmers Guide to the E Editor Toolkit.* 1993.

Pedgen,C. D. and D. A. Davis. 1992. *Arena^{TM}: A SIMAN\CINEMA Based Hierarchical Modelling System*, Proceedings of the 1992 Winter Simulation Conference.

B. P. Zeigler. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models.* Academic Press, 1990.

AUTHOR BIOGRAPHIES

THOMAS BEHLAU works as a graduate research assistent in the Department of Computer Simulation and Graphics at the University of Magdeburg. His areas of research are the modeling of manufacturing systems and modeling methodologies. He is a member the GPSS-Users'-Group Europe.

VOLKMAR HINZ works as a Assistent Professor in the Department of Computer Simulation and Graphics at the University of Magdeburg. His research interests are focused on environments of simulation systems. He is a member of the GPSS-Users'-Group Europe.