HGPSS:
Hierarchical General Purpose Simulation System

User Manual

Filip Claeys
Supervisor: Hans Vangheluwe

1992
## Contents

1 Introduction 4

2 A solution to the GPSS weaknesses 6

3 The HGPSS language 7
   3.1 General form of a HGPSS program 7
   3.2 Labels 8
   3.3 Comments 8
   3.4 Embedded C++ code 8
   3.5 Referencing C++ variables and functions in HGPSS 9
   3.6 Functions and variables 9
   3.7 Referencing HGPSS entities in C++ code 9
   3.8 Hierarchical modeling 11
   3.9 Parameterised models 13
   3.10 Interfacing and extending the language 13
   3.11 Multiple analogous declarations 15
   3.12 Output 15

4 HGPSS versus GPSS 18
   4.1 Introduction 18
   4.2 Block declarations 18
      4.2.1 GPSS block declarations 18
      4.2.2 HGPSS block declarations 20
   4.3 Entity declarations 21
      4.3.1 GPSS entity declarations 21
      4.3.2 HGPSS entity declarations 22
   4.4 Commands 22
      4.4.1 GPSS commands 22
      4.4.2 HGPSS commands 22

5 How to create a simulator 23
   5.1 HGPSS system files 23
   5.2 The HGPSS to HGPSS++ compiler 24
   5.3 Compiling and linking the simulator 24

6 Extended Backus Naur Form description 25

7 Statement summary 30
List of Figures

3.1 A self-service shopping system .................................................. 7
3.2 The shopping system extended with C++ code ........................................ 9
3.3 The shopping system with reference to a C++ variable .......................... 10
3.4 The GPSS and HGPSS way of function and variable declaration ............... 10
3.5 A multi-model system .................................................................. 11
3.6 Model hierarchy of previous figure ...................................................... 11
3.7 A multi-model system without transaction transfer ............................... 12
3.8 Model hierarchy of previous figure ...................................................... 12
3.9 The UP and DOWN commands .......................................................... 13
3.10 Process model of the HGPSS system ................................................ 14
3.11 The shopping system with a clock .................................................... 15
3.12 The shopping system using an INTERN block ..................................... 16
3.13 The shopping system using an EXTERN block .................................... 16
3.14 Low-level actions using an INTERN block ......................................... 16
3.15 Multiple analogous declarations ...................................................... 17
3.16 Some output related commands ...................................................... 17
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Metalinguistic variables</td>
<td>31</td>
</tr>
<tr>
<td>7.2</td>
<td>HGPSS Standard Numerical Attributes</td>
<td>32</td>
</tr>
<tr>
<td>7.3</td>
<td>HGPSS entity mnemonics</td>
<td>33</td>
</tr>
<tr>
<td>7.4</td>
<td>HGPSS block declaration summary, part 1</td>
<td>34</td>
</tr>
<tr>
<td>7.5</td>
<td>HGPSS block declaration summary, part 2</td>
<td>35</td>
</tr>
<tr>
<td>7.6</td>
<td>HGPSS block declaration summary, part 3</td>
<td>36</td>
</tr>
<tr>
<td>7.7</td>
<td>HGPSS entity declaration summary</td>
<td>37</td>
</tr>
<tr>
<td>7.8</td>
<td>HGPSS command summary</td>
<td>38</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The discrete-event simulation language GPSS (or General Purpose Simulation System) was originally developed by Gordon at IBM in 1962. The language has been used extensively and different commercial implementations were developed ever since its conception. Despite its expressive power and flexibility, the language suffers from some weaknesses. Three major deficiencies can be distinguished:

- Lack of conceptual support for hierarchical modeling
- Poor interfacing facilities with other software components
- No possibility of interfering with the way the GPSS processor works

By means of the HGPSS project, an effort has been made to resolve these three problems. The resulting HGPSS language can be regarded as a superset of the GPSS/360 dialect, introduced in 1967. In the following sections, no attempt will be made to clarify the basics of the GPSS/360 language. In depth information can be found in [Schriber 1974], [Neelamkavil 1987] and [Gordon 1978].

All three major GPSS problems have been resolved by the implementation of a two-level system.

- The bottom level consists of a host language object-oriented programming environment. A discrete-event simulation kernel has been developed in this host language. The kernel consists of a number of interrelated functions and classes for the support of GPSS-like process-interaction simulation. Calls to this kernel are implemented by a multitude of interface functions. This interface language has been denominated HGPSS++. It should be noted that performance issues were not taken into account in this stage of the development of the HGPSS++ kernel.

- The top level is made up of a compiler that converts HGPSS program statements into a set of corresponding HGPSS++ kernel calls.

C++ has been chosen as the host language for the first level of the system. The C++ language is especially well suited for discrete-event simulation. The object-oriented philosophy was conceived with the development of the discrete-event simulation language SIMULA, the early forerunner of C++. An elaborate description of the C++ language can be found in [Stroustrup 1987].

The HGPSS to HGPSS++ compiler has been implemented with the use of the lexical analyser generator LEX and the parser generator YACC [Mason & Brown 1990].

The bottom level simulation kernel and the HGPSS++ language will not be discussed any further. Additional information can be found in [Claeys 1992]. The top level however will be discussed in detail.

One might wonder why GPSS, despite its disadvantages, is still interesting as a basis for a new discrete-event simulation system. The set of additional features offered by GPSS in respect to other simulation languages may serve as an answer to this question. The most important advantages of GPSS can be categorised as follows:

- The process-interaction world view.
- The fact that the system is very well-know and widely used.
• The high-level character of the language constructs.

In fact, GPSS is a full programming language [Boehm & Jacopini 1966], because it offers
• Basic actions,
• Action sequences,
• Conditional structures,
• Iterative structures.
Chapter 2

A solution to the GPSS weaknesses

The first goal of the HGPSS project, namely to provide some conceptual support for hierarchical modeling, has been accomplished by the introduction of the submodel concept, reflected in the introduction and implementation of four additional basic simulation blocks:

- ENTERMODEL
- LEAVEMODEL
- INPUT
- OUTPUT

A submodel is considered as an entity and has to be declared by the newly implemented SUBMODEL declaration statement. The idea is to break up the model of the system to be simulated into a number of submodels. Each submodel can in turn be subdivided into a number of submodels. By applying this procedure recursively, a tree-like model hierarchy will evolve. In the end, the submodels represented by the leaves of the tree should represent the basic processes within the system. Furthermore, such a hierarchical decomposition approach supports stepwise refinement, in which atomic blocks, in particular tree leaves, can be arbitrarily decomposed to reflect more detailed, microscopic knowledge about the system.

The second goal, the creation of a method of interfacing with other software components, has been implemented in a number of ways. Constructs of the C++ host language can be interlaced with HGPSS statements in many ways. Two additional basic simulation blocks also provide interfacing support. These blocks are:

- INTERN
- EXTERN

By letting the modeler interfere with the way the HGPSS processor works, the final goal of the project is accomplished. The HGPSS processor is designed as an open system that can be expanded and modified in many ways. See [Claeys 1992] for further details on this subject.
Chapter 3

The HGPSS language

3.1 General form of a HGPSS program

A HPGSS program, unlike a GPSS program, consists of a number of separate sections. There are two kinds of sections: model sections and command sections. Each program can only contain one command section. There is no limitation to the number of model sections. The presence of the command section is not mandatory. The number of model sections can be zero as well. The model sections can be regarded as the model frame of the system, whereas the command section represents the experimental frame [Zeigler 1976]. Each of the model sections represents one of the submodels the system is made out of. The system is modeled entirely by the collection of models and their interrelations. The command section depicts the environment the system is in. In general, the command section will specify what the inputs to the system are and what kind of output the modeler is interested in.

The following simple example will illustrate the concept of model and command section. A single-server system, more specifically a self-service shop with one cash desk, will be considered. Customers enter the shop, take the goods they want, then join the queue of customers waiting to check out and finally leave the queue and pay the bill. Figure 3.1 shows the HPGSS program for simulation of this system. The line numbers are not part of the language and are simply used for referencing. The HPGSS program consists of one model section (lines (1)-(10)) and one command section (lines (12)-(16)).

A model section always has to be encapsulated between a MODEL and an ENDMODEL statement. The MODEL keyword has to be followed by the name of the model. The command section begins with COMMAND and ends with ENDCOMMAND. Each HGPSS statement takes exactly one line. There are four kinds of statements:

- Block declaration statements
- Entity declaration statements

```
(1) MODEL Shop
(2) GENERATE 10,7 Enter the shop
(3) ADVANCE 20,8 Take goods
(4) QUEUE Wait Join queue
(5) SEIZE CashDesk Arrive at cash desk
(6) DEPART Wait Leave the queue
(7) ADVANCE 8,2 Pay the bill
(8) RELEASE CashDesk Leave the cash desk
(9) TERMINATE 1 Leave the shop
(10) ENDMODEL

(11) COMMAND
(12) SIMULATE Shop Install model
(13) START 1000 Simulate 1000 customers
(14) PRINT / Print all
(15) END Remove model
(16) ENDCOMMAND
```

Figure 3.1: A self-service shopping system
The last category includes only four statements:

- MODEL
- ENDMODEL
- COMMAND
- ENDCOMMAND

The command section consists entirely of command statements. The model sections are constructed by mixing block declaration statements with entity declaration statements. A complete summary of all HGPSS statements except the last category can be found in Tables 7.4, 7.5, 7.6, 7.7 and 7.8. Each statement can be prepended with white space and does not have to start in the first position of the line as in GPSS. Contrary to GPSS, HGPSS is no longer FORTRAN-based. Block and entity declarations can optionally be labeled. A label must precede the statement keyword and must be separated from the rest of the statement by white space. An exact definition of white space can be found in the extended Backus Naur Form description of the HGPSS language. Each HGPSS statement consists of a keyword followed by a list of parameters. The parameters must be separated from the keyword by white space. The parameters are separated from one another by a comma. There is no white space allowed between parameters and commas. The number and type of the parameters depends upon the statement. The parameter list can optionally be followed by a comment, separated from this parameter list by white space. Consider for instance line (2). The line contains a block declaration statement. The keyword is GENERATE and the parameters are respectively 10 and 7. ‘‘Enter the shop’’ is a comment.

### 3.2 Labels

In GPSS, labels where restricted to capitals and a maximum of 5 characters. In HGPSS, labels can be up to 50 characters long. They may contain any letter or digit, and also underscores and dollar-signs. Labels with a digit as a leading character are not allowed. One must take care not to use labels that clash with HGPSS reserved words. Since HGPSS reserved words always consist of nothing but capitals, the clashing problem will certainly not arise when only lower case labels are used. Lower case labels will also improve program readability.

### 3.3 Comments

In Figure 3.1, one way of writing comments is illustrated. In fact there is another way to write comments in addition to putting them behind the statement parameters. Every source line that has a * as a first non-white space character, is considered as a line of comments.

### 3.4 Embedded C++ code

One of the major advantages of the HGPSS language is the possibility of writing C++ code between the HGPSS statements. The C++ code in a HGPSS program is simply copied into the file containing the generated HGPSS++ kernel calls, to be processed by the C++ compiler.

Figure 3.2 is an extension to the shopping system and illustrates the ways of putting C++ code into a HGPSS program. Line (1) shows the first way to put C++ code into a program. This method is very much like prepending a line of comments with a *. In the case of C++ code, the line has to be prepended with a -. Anything following the – until the end of the line will be passed literally to the HGPSS++ translator. Lines (5)-(9) show another way of putting C++ code into a program. A block of C++ code has been encapsulated within a %{ -%} pair. The % has to be the first non-white space character sequence on the line. The % sequence on the other hand has to be the very last non-blank token on the line. It is obvious that the combination of % and % will be the best way to declare a large block of C++ code, while the – will be better in case there is just one line to be declared as such.
```c++
#include <iostream.h>

MODEL Shop

{% /* The following line of C++ code will output a message to the standard output to indicate that the model section is being processed simulation starts */

cout << "Model section being processed" << "\n";
%

GENERATE 10,7 Enter the shop
ADVANCE 20,8 Take goods
QUEUE Wait Join queue
SEIZE CashDesk Arrive at cash desk
DEPART Wait Leave the queue
ADVANCE 8,2 Pay the bill
RELEASE CashDesk Leave the cash desk
TERMINATE 1 Leave the shop
ENDMODEL

COMMAND
SIMULATE Shop Install model
START 1000 Simulate 1000 customers
PRINT / Print all
END Remove model
ENDCOMMAND
```

Figure 3.2: The shopping system extended with C++ code

### 3.5 Referencing C++ variables and functions in HGPSS

Since C++ code is allowed in between HGPSS statements in a program there must also be a way of referencing C++ variables and functions in a pure HGPSS statement. This can be accomplished by double quoting calls to C++ variables and functions in HGPSS statements. This is illustrated in Figure 3.3. In line (11) there is a reference being made to the variable `spread`. In fact not only variables and function calls but also entire C++ expressions can be put between double quotes. Lines (4) and (5) could for instance be deleted from the previous example and line (11) replaced by `ADVANCE 8, "6+sqrt(2)"`.

### 3.6 Functions and variables

The GPSS FUNCTION, VARIABLE, BVARIABLE and FVARIABLE statements are supported by HGPSS, but these statements have been extended. The extension makes it possible to let the entities declared by the previous statements refer to C++ functions. These C++ functions must have `value_type` (in the case of the FUNCTION, VARIABLE and FVARIABLE statements) or `boolean_type` (in the case of the BVARIABLE statement) as the type of the returned values. These types are known to the HGPSS++ translator.

In Figure 3.4, the GPSS and HGPSS way of function and variable declaration are illustrated. In line (18) the C++ function `CppVar` is assigned to the HGPSS variable `HGPSSVar`. In line (30), the C++ function `CppFun` is assigned to the HGPSS function `HGPSSFun`. In both cases the name of the C++ function is double-quoted. The slash in front of the name of the function is mandatory to avoid confusion with the usual way of declaring variables and functions. Especially when functions are concerned, a lot can be gained by applying the HGPSS way. It makes the use of vast function lookup tables obsolete.

In Figure 3.4, another interesting feature is displayed. In lines (5) and (8), a random number generator other than the built-in HGPSS random number generators is used, i.e. the function `rand()`, which is part of the C++ standard library.

### 3.7 Referencing HGPSS entities in C++ code

Referencing C++ variables and functions in HGPSS statements is easy as illustrated before. One can also do the opposite: reference HGPSS entities in C++ code. This however requires the use of HGPSS++ kernel calls. Refer to [Claeys 1992] for more details.
Figure 3.3: The shopping system with reference to a C++ variable

Figure 3.4: The GPSS and HGPSS way of function and variable declaration
Perhaps the most interesting extension to the GPSS language is the provision of some hierarchical modeling constructs. In all previous examples, only single-model programs were considered. A program can however contain several model sections, each describing a submodel of the complete model of the system. One of the model sections must describe the model at the root of the hierarchy tree. The name of this model must be notified to the system by means of the `SIMULATE` command. Each submodel used in another model must be declared by the `SUBMODEL` statement. A transaction currently in a certain model can be routed to a submodel relative to the first model by the use of the `ENTERMODEL` block. This block takes two parameters. The first parameter refers to the submodel the transaction is to be routed to. The second parameter refers to the location in the submodel where the transaction is to enter. These locations can be declared by the `INPUT` block. The `INPUT` block takes as a parameter the name of the input. The locations where a transaction can return to the model it is originating from, are marked by means of an `OUTPUT` block. This block takes as a parameter the name of the output. Lastly there is the `LEAVEMODEL` block. This block marks the location where the transaction is to return in the model it is originating from. A `LEAVEMODEL` block takes two parameters. The first one refers to the submodel and the second one to the output of the submodel.

Figure 3.5 will clarify the mechanism. The goal is to construct a program that describes a system consisting of two subsystems. The second subsystem is to be activated after the first subsystem has ended its task. The situation of Figure 3.5 can also be depicted by Figure 3.6. The lines in the figure without an arrow refer to the actual hierarchy, whereas the arrowed lines refer to the transaction flow. It is important to distinguish hierarchy from transaction flow}

3.8 Hierarchical modeling


Figure 3.5: A multi-model system

![Figure 3.5: A multi-model system](image)

Figure 3.6: Model hierarchy of previous figure

![Figure 3.6: Model hierarchy of previous figure](image)
because transactions need not necessarily flow from model to submodel and back.

In Figure 3.7, the situation is described where a model consists of two submodels without transfer of transactions. In fact this program models a system consisting of two subsystems that operate *concurrently* and without interaction. This situation can as in the previous example also be depicted by a diagram as in Figure 3.8.

All kinds of combinations of the previous two situations are possible. In principle there is no limitation to the depth of the hierarchy tree.

Each model has its own context. All entities used in a model are local. The only means of interchanging information between models are:

- via the transfer of transactions,
- via global C++ variables.

Entities can only be referenced from within the command section one model at a time, this because there is only one model active in the command section at a time. Initially the active model is the root model specified in the SIMULATE command. The active model can be changed to one of its submodels by using the DOWN command. This command allows descending the hierarchy tree and takes as a parameter the name of the submodel to go to. Only when the active model is the root model, identifier-labels can be used. In the other cases number-labels must be used. One can go up the hierarchy tree by the UP command. This command takes no parameters.

Figure 3.9 presents some examples of the use of the UP and DOWN commands. The command in line (22) will cause all storages of the root model to be printed, whereas the command in line (24) will produce some output on the storages of the root model’s submodel named ModelA. In line (25), the use of a numeric label is mandatory.

When using identifier-labels as respective input name and output name parameter in the ENTERMODEL and LEAVEMODEL statements, it is necessary to declare the submodel by the SUBMODEL statement before the first occurrence of the ENTERMODEL or LEAVEMODEL statement. The compiler must know what kind of submodel is being referred to in these two statements.

All submodels used in a model need not be described in the same program file. It is possible to compile submodels separately and link them with other submodels and the command section. In Figure 3.10, an overview of the HGPSS
3.9 Parameterised models

Models can be used as templates that will be adapted to a certain situation by the use of parameters. The way of assigning parameters to models works the same way as using function parameters in C++.

Figure 3.11 shows how a parameterised model can be constructed and used. The shopping system is considered again. The goal is not to simulate a thousand customers as in the previous cases but 24 hours. A clock model is therefore constructed that will signal the expiring of the time span. At line (14) the MODEL reserved word is followed by the name of the parameterised model which is on its turn followed by the formal parameter list. This starts with a ( and ends with a ). Anything between these two characters will be regarded as the C++ description of the parameter list. At line (2) the Clock model is declared as a submodel relative to the Shop model. The name of the submodel is followed by the actual parameter list.

It is important to note that there is only one termination counter in the whole HGPSS system. Each TERMINATE block on whatever level of the model hierarchy will consequently refer to the same termination counter.

3.10 Interfacing and extending the language

A way of extending the HGPSS language and interfacing with other software components is by making use of the INTERN and EXTERN blocks.

The INTERN block takes as a parameter the name of a C++ function that will be triggered when a transaction enters the block.

The EXTERN block takes two parameters. The first parameter is the name of the C++ function that will be triggered when a transaction enters the block. The second parameter is the name of a C++ function that will be triggered on every active simulation time point. It is clear that the EXTERN block provides a way of interfacing with other programs or routines that are asynchronous to the discrete-event simulation taking place.
Figure 3.10: Process model of the HGPSS system
(1) MODEL Shop
(2) 1 SUBMODEL Clock(24) Initialize clock
(3)
(4) GENERATE 10,7 Enter the shop
(5) ADVANCE 20,8 Take goods
(6) QUEUE Wait Join queue
(7) SEIZE CashDesk Arrive at cash desk
(8) DEPART Wait Leave the queue
(9) ADVANCE 8,2 Pay the bill
(10) RELEASE CashDesk Leave the cash desk
(11) TERMINATE / Leave the shop
(12) ENDMODEL
(13)
(14) MODEL Clock(long time)
(15) GENERATE "time*60" 
(16) TERMINATE 1
(17) ENDMODEL
(18)
(19) COMMAND
(20) SIMULATE Shop Install model
(21) START 1 Simulate model
(22) PRINT / Print all
(23) END Remove model
(24) ENDCOMMAND

Figure 3.11: The shopping system with a clock

Figure 3.12 presents an example of the use of an INTERN block. Each time a transaction reaches the block, a message will be printed on the standard output.

In Figure 3.13, a message is printed at every active simulation time point. The EXTERN block is put outside the transaction flow because there is no relevant action to be performed if a transaction were to enter the block. Consequently the NULL refers to the fact that there is no function to be called upon arrival of a transaction.

Figure 3.14 illustrates the use of an INTERN block to perform low-level actions on the transactions entering the block. The low-level actions are carried out by HGPSS++ kernel calls.

### 3.11 Multiple analogous declarations

Suppose several entities of the same kind have to be declared. Instead of writing the desired number of declaration statements, a procedure as illustrated in Figure 3.15 can be used. Multiple copies of a storage with capacity 2 are created by putting the STORAGE command inside a loop. A C++ variable is used as a label. Note that the use of C++ expressions as labels is only allowed in the context of the declaration of entities and not blocks. This method does not work when the entities have to be identifier-labeled.

### 3.12 Output

A GPSS simulation automatically generates an output file containing all information gathered during the simulation process. The HGPSS system on the other hand does not output information unless explicitly asked for. The information relevant for a certain simulation can be selected. In fact, there are two ways of extracting information from the system. First of all there is the PRINT block. This block also exists in GPSS. The parameters of the HGPSS block however are somewhat different from those of the GPSS block. The first and second parameter are the same as in GPSS. They are also optional. The third parameter is as in GPSS an entity mnemonic. The mnemonics are not the same as in GPSS. Table 7.3 lists all HGPSS entity mnemonics. When this parameter is unspecified, information about all entities will be output. The optional fourth HGPSS parameter is a C++ pointer to a file. When this parameter is unspecified, information will be sent to the default output file. When the parameter is specified, information can be redirected to the file pointed to by the C++ file pointer.

Another way to generate output is by using the PRINT command. This command works entirely the same way as the block with the same name.

By default, the name of the output file is hgpsspp.out. The default file name can be changed by giving the new name as a second parameter to the SIMULATE command.
#include <iostream.h>

void Message(void) {
    cout << "A customer pays the bill\n";
}

MODEL Shop
GENERATE 10,7 Enter the shop
ADVANCE 20,8 Take goods
QUEUE Wait Join queue
SEIZE CashDesk Arrive at cash desk
DEPART Wait Leave the queue
INTERN \"Message\" Print message
ADVANCE 8,2 Pay the bill
RELEASE CashDesk Leave the cash desk
TERMINATE 1 Leave the shop

ENDMODEL

COMMAND
SIMULATE Shop Install model
START 1000 Simulate 1000 customers
PRINT / Print all
END Remove model
ENDCOMMAND

Figure 3.12: The shopping system using an INTERN block

MODEL Shop
EXTERN \"NULL\",\"Message\"
...
INTERN \"Copy\"
...
ENDMODEL

Figure 3.13: The shopping system using a EXTERN block

// The following function copies the number of parameters of
// the active transaction into its first parameter.
void Copy(void) {
    TransactionClass *transaction;
    transaction=Processor.Transaction();
    transaction->setParameter(1,transaction->nbrOfParameters());
}

MODEL Model
INTERN \"Copy\"
...
ENDMODEL

Figure 3.14: Low-level actions using an INTERN block
Figure 3.15: Multiple analogous declarations

```plaintext
MODEL Model
*Writing the same statement 5 times in GPSS,
(5) 1 STORAGE 2
(6) 2 STORAGE 2
(7) 3 STORAGE 2
(8) 4 STORAGE 2
(9) 5 STORAGE 2
*reduces to this in HGPSS
(10) for (int i=1; i<=5; i++)
    "i" STORAGE 2
ENDMODEL
```

Figure 3.16: Some output related commands

```plaintext
#include <fstream.h>
COMMAND
ofstream file_pointer;
file_pointer=fopen("output.txt");
*Output info on all facilities to default output file
PRINT ,,FACILITY
*Output info on facility A to default output file
PRINT A,,FACILITY
*Output info on absolute clock to file pointed to by "file_pointer"
PRINT ,,ABSOLUTECLOCK,"file_pointer"
*Change name of default output file to "output.txt"
SIMULATE ModelName,"output.txt"
ENDCOMMAND
```

Figure 3.16 presents some examples of output related commands.
Chapter 4

HGPSS versus GPSS

4.1 Introduction

In this section, an overview of the syntactic and semantic differences between GPSS/360 and HGPSS will be presented. Also, the new block declarations, entity declarations, and commands will be discussed.

4.2 Block declarations

4.2.1 GPSS block declarations

1. ADVANCE:
   - Syntax: When all parameters are omitted, a slash is needed to eliminate some parser ambiguities.
   - Semantics: No differences.

2. ASSEMBLE: No differences.

3. ASSIGN: No differences.

4. BUFFER: No differences.

5. DEPART: No differences.

6. ENTER: No differences.

7. GATE: No differences.

8. GATHER: No differences.

9. GENERATE:
   - Syntax: When all parameters are omitted, a slash is needed to eliminate some parser ambiguities.
   - Semantics:
     - The optional parameter which indicates the type of the parameters of the transactions generated, has no effect. Whether this parameter is supplied or not, the type of the parameters will always be value_type.
     - The number of parameters a transaction will be supplied with has to be specified explicitly. Defaulting the number of parameters is allowed, but results in the generation of a transaction with zero parameters.

10. LEAVE: No differences.

11. LINK: No differences.
12. LOGIC: No differences.
13. LOOP: No differences.
14. MARK:
   • Syntax: When all parameters are omitted, a slash is needed to eliminate some parser ambiguities.
   • Semantics: No differences.
15. MATCH: No differences.
16. MSAVEVALUE:
   • Syntax: No differences.
   • Semantics: The optional matrix type parameter has no effect. Whether this parameter is supplied or not, the type of the elements of the matrix will always be value_type.
17. PREEMPT:
   • Syntax: No differences.
   • Semantics: In GPSS there are three alternatives to the way a transaction that has been preempted from a facility, behaves:
     (a) The preempted transaction is put on an interrupt chain and will be restored to the facility when the facility is returned to it.
     (b) The preempted transaction is routed to some other block in the model, and is removed from contention for use of the facility. That is, it will not later be automatically reinstated by the processor as the capturer of the facility.
     (c) The preempted transaction is routed to some other block in the model, but is not removed from contention for later automatic reinstatement as the capturer of the facility.
     In HGPSS only the first alternative is implemented. The parameters of the PREEMPT statement that refer to the two other alternatives can however be supplied but will have no effect.
18. PRINT:
   • Syntax:
     – When all parameters are omitted, a slash is needed to eliminate some parser ambiguities.
     – The third parameter still refers to an entity mnemonic but the HGPSS entity mnemonics are different to those of GPSS.
     – There is an optional fourth parameter. This parameter refers to a C++ file pointer. The output generated by the block will be sent to the file pointed to by the pointer, and not to the default output file.
   • Semantics: No differences.
19. PRIORITY: No differences.
20. QUEUE: No differences.
21. RELEASE: No differences.
22. RETURN: No differences.
23. SAVEVALUE:
   • Syntax: No differences.
   • Semantics: The optional savevalue type parameter has no effect. Whether this parameter is supplied or not, the type of the savevalue will always be value_type.
24. **SEIZE**: No differences.

25. **SELECT**: No differences.

26. **SPLIT**: No differences.

27. **TABULATE**: No differences.

28. **TERMINATE**:
   - Syntax: When all parameters are omitted, a slash is needed to eliminate some parser ambiguities.
   - Semantics: No differences.

29. **TEST**: No differences.

30. **TRANSFER**:
   - Syntax: In the case of a statistical block and a Standard Numerical Attribute as the selection mode parameter, the selection mode parameter had to be prepended with a dot in GPSS. This dot was required to indicate the fact that the integer value defined by the selection mode parameter was considered as the fractional part of a real with zero as the integer part. In HGPSS however this parameter should by itself represent a real between zero and one. The dot is consequently not allowed in HGPSS.
   - Semantics: No differences.

31. **UNLINK**: No differences.

### 4.2.2 HGPSS block declarations

1. **ENTERMODEL**: When a transaction enters this block, it will be removed from the model it is currently in, and will be injected in the submodel specified by the first parameter. The second parameter is the number of the input of the submodel the transaction is injected into.

2. **EXTERN**: This block allows interfacing with other programs or routines that are asynchronous to the discrete-event simulation. The block takes two parameters. The first parameter is the name of a C++ function that will be called when a transaction enters the block. The second parameter is the name of a function that will be called on every active simulation time point. To indicate that there is no function to be called, a **NULL** may be entered as a parameter.

3. **INPUT**: This block indicates an input to a model. Via this input, transactions originating from a higher level in the model hierarchy can be injected in the model. The parameter specifies the name of the input.

4. **INTERN**: This block allows interfacing with other routines or programs. The block takes one parameter. This parameter is the name of a C++ function that will be called when a transaction enters the block.

5. **LEAVEMODEL**: This block indicates the location in the model where a transaction that has previously been injected in a submodel, can return into the first model. The first parameter specifies the submodel and the second parameter the number of the output of the submodel.

6. **OUTPUT**: This block indicates an output to the model. Via this output, transactions originating from a higher level in the model hierarchy can return to the model they originated from. The parameter specifies the number or name of the output.
4.3 Entity declarations

4.3.1 GPSS entity declarations

1. BVariable:
   - Syntax: Besides the original GPSS form of the statement, there is another newly implemented form. The HGPSS boolean variable can be assigned to a C++ function. Each time the boolean variable is referenced, the C++ function will be called. The name of the C++ function has to be double-quoted and prepended with a slash.
   - Semantics: No differences.

2. Function:
   - Syntax:
     – In case the list of function points is spread over more than one line, this has to be indicated with a – at the end of all lines but the last.
     – Besides the original GPSS form of the statement, there is another newly implemented form. The HGPSS function can be assigned to a C++ function. Each time the function is referenced, the C++ function will be called. The name of the C++ function has to be double-quoted and prepended with a slash.
     – As opposed to GPSS, matrix Standard Numerical Attributes are allowed as function arguments.
   - Semantics: No differences.

3. Matrix:
   - Syntax: No differences.
   - Semantics: The matrix type parameter has no effect. Whether this parameter is set to H or F, the type of the elements of the matrix will always be value_type.

4. QTable: No differences.

5. Storage:
   - Syntax: Only one of the two GPSS forms of storage declaration is allowed.
   - Semantics: No differences.

6. Table: No differences.

7. Variable:
   - Syntax: Besides the original GPSS form of the statement, there is another newly implemented form. The HGPSS variable can be assigned to a C++ function. Each time the variable is referenced, the C++ function will be called. The name of the C++ function has to be double-quoted and prepended with a slash.
   - Semantics: No differences.

8. FVariable:
   - Syntax: Besides the original GPSS form of the statement, there is another newly implemented form. The HGPSS fullword variable can be assigned to a C++ function. Each time the fullword variable is referenced, the C++ function will be called. The name of the C++ function has to be double-quoted and prepended with a slash.
   - Semantics: There is no difference between variables and fullword variables. Both will have to return a value_type as a result.
4.3.2 HGPSS entity declarations

1. SUBMODEL: This statement declares a model as a submodel relative to another model. The declaration takes as a parameter the type of the submodel. It is favourable always to put the SUBMODEL statement at the top of a model section. By doing this, the HGPSS system will always know the type of the submodels used.

4.4 Commands

4.4.1 GPSS commands

1. CLEAR:
   - Syntax: There are no parameters allowed to this statement, contrary to GPSS. As a result there is no way to retain the contents of savevalues after a CLEAR command is carried out. C++ variables can however always be used to retain information.
   - Semantics: No differences.

2. END: No differences.

3. INITIAL:
   - Syntax:
     – Multiple entries separated by a slash are not allowed.
     – The use of - to indicate subranges is not allowed.
   - Semantics: No differences.

4. JOB: No differences.

5. RESET: No differences.

6. SIMULATE:
   - Syntax: This command has the same name as the GPSS SIMULATE command but has nothing in common with this command. The command is used to tell the system by means of the first parameter which model is located at the root of the hierarchy tree. The second parameter is optional and allows the default output file name to be overridden.
   - Semantics: See syntax.

7. START:
   - Syntax: The printout suppression, initial value of snap interval counter and signal for chain printouts parameters are not supported.
   - Semantics: No differences.

4.4.2 HGPSS commands

1. DOWN: This command allows descending the hierarchy tree. The only parameter refers to the submodel that has to activated.

2. PRINT: This command works entirely the same way as the PRINT block.

3. UP: This command allows climbing the hierarchy tree. The command uses no parameters.
Chapter 5

How to create a simulator

In this section, all the necessary steps to create a simulator will be presented. First of all, the files the HGPSS system consists of are listed.

5.1 HGPSS system files

- HGPSS to HGPSS++ compiler:
  - hgpss.l: LEX description of the lexical analyser for use on UNIX systems.
  - hgpss.lxi: LEX description of the lexical analyser for use on MSDOS systems.
  - hgpss.y: YACC description of the parser. There are two versions of this file: one for use on UNIX systems and one for use on MSDOS systems.
  - initial.y: YACC description of auxiliary initial parameter parser.
  - logical.y: YACC description of auxiliary logical parameter parser.
  - param.y: YACC description of auxiliary parameter parser.
  - tokens.h: List of all tokens for use with hgpss.l or hgpss.lxi.
  - hgpss.exe: Executable HGPSS to HGPSS++ compiler for use on MSDOS systems.
  - hgpss: Executable HGPSS to HGPSS++ compiler for use on UNIX systems.

- HGPSS++ kernel:
  - Access files:
    * hgpsspp.h: Header file granting partial access to the HGPSS++ kernel. This file will be included in compiled HGPSS programs.
    * hgpssppa.h: Header file granting complete access to the HGPSS++ kernel. This file will be included upon request in compiled HGPSS programs.
  - System files:
    * header.h: Header file to be included in all HGPSS++ system files.
    * funcsup.C: Support functions.
    * funcuser.C: User functions.
    * classsup.C: Support classes.
    * classsys.C: System classes.
    * classent.C: Entity classes.
    * classenc.C: Entity chain classes.
    * classblo.C: Block classes.
    * classpro.C: Processor classes.
5.2 The HGPSS to HGPSS++ compiler

Once a HGPSS program is written, it should be compiled to a HGPSS++ program with the HGPSS to HGPSS++ compiler. This program generates three files from the input file:

1. A so called output file, which is the actual HGPSS++ program.
2. A so called header file, containing a class declaration and a constant structure containing the resolved labels for all the models in the input file.
3. A so called variable file, containing C++ functions automatically generated from GPSS-like VARIABLE, BVARIABLE and FVARIABLE statements.

The two last files will be automatically included in the first one when the HGPSS++ program is compiled with a C++ compiler.

The HGPSS to HGPSS++ compiler needs some command line arguments. There are four possible forms these command line arguments can take:

1. `-e name`
   - The file hgpsspp.h will be included in the output file.
   - The input file name.gps will be processed.
   - The output file will be denominated name.C.
   - The header file will be denominated name.h.
   - The variable file will be denominated name.var.

2. `-f [name1 [name2 [name3 [name4]]]]`
   - The file hgpsspp.h will be included in the output file.
   - The input file name1 will be processed. The default input file name is hgpsspp.gps.
   - The output file will be denominated name2. The default output file is hgpsspp.C.
   - The header file will be denominated name3. The default header file is hgpsspp.h.
   - The variable file will be denominated name4. The default variable file is hgpsspp.var.

3. `-E name`
   - The file hgpssppa.h will be included in the output file.
   - The input file name.gps will be processed.
   - The output file will be denominated name.C.
   - The header file will be denominated name.h.
   - The variable file will be denominated name.var.

4. `-F [name1 [name2 [name3 [name4]]]]`
   - The file hgpssppa.h will be included in the output file.
   - The input file name1 will be processed. The default input file name is hgpsspp.gps.
   - The output file will be denominated name2. The default output file is hgpsspp.C.
   - The header file will be denominated name3. The default header file is hgpsspp.h.
   - The variable file will be denominated name4. The default variable file is hgpsspp.var.

5.3 Compiling and linking the simulator

After the HGPSS++ output, header and variable file have been generated out of the HGPSS input file by the HGPSS to HGPSS++ compiler, the output file should be compiled with a C++ compiler. The object file resulting from the output file should then be linked with the object files resulting from the compilation of the HGPSS++ system files.
Extended Backus Naur Form description

This section contains an Extended Backus-Naur Form description of the HGPSS-language. The notation consists of metalinguistic variables, metalinguistic symbols, HGPSS reserved words and HGPSS symbols. The grammar itself is presented as a sequence of rules, where each rule consists of a left-hand side and a right-hand side connected by the metasymbol ::= The left-hand side of the rule is a metavariable. The right-hand side of the rule is an expression consisting of metavariables, metasymbols, HGPSS reserved words and HGPSS symbols. Wherever a metavariable appears in the right-hand side of a rule, the right-hand side of any rule having that variable as its left-hand side can be unambiguously substituted for the variable. This is the distinguishing characteristic of context-free grammars. A metavariable is a character string consisting of letters and underscores. The variable name is chosen to be meaningful in its context. The metasymbols to be used are the following:

- ::= connects left- and right-hand side of a rule.
- | separates alternative right-hand side items.
- [] encloses an optional right-hand side item.
- {} encloses a repeated right-hand side item that can appear zero or more times.
- () is used for grouping alternative right-hand side items.
- - indicates a class of characters.

HGPSS reserved words are distinguished from metavariables by consisting only out of capitals. Exceptions to this rule are single quoted. HGPSS symbols are not quoted, except when there might be confusion with metasymbols. Other symbols used:

- \t tabulate character.
- \r line feed character.
- \n new line character.

program ::= model_sections command_section
model_sections ::= { model_section | start text }
model_section ::= start MODEL white_space identifier [ '(' anything_except_ ')'] end decls
                start ENDMODEL end
decls ::= { start decl }
decl ::= block_decl_body end
| identifier white_space block_decl_body end
| integer white_space entity_decl_body end
| expression white_space entity_decl_body end
| identifier white_space entity_decl_body end
| text
command_section ::= { start COMMAND end
| commands
| start ENDCOMMAND end
| texts }
commands ::= { start command }
command ::= command_body end

texts ::= { start text }

text ::= %{ anything_except_%} %} new_line
| - anything_except_new_line new_line
| * anything_except_new_line new_line
| new_line

block_decl_body ::= ADVANCE white_space ( /
| parameter
| [ parameter ] , parameter |
| ASSEMBLE white_space parameter
| ASSIGN white_space parameter [ + | - ] , parameter [ , parameter ]
| BUFFER white_space parameter [ , parameter ]
| ENTER white_space parameter [ , parameter ]
| ENTERMODEL white_space ( expression , ( integer | expression )
| ( integer | identifier ) , entity_name )
| EXTERN white_space expression , expression
| GATE white_space gate_kind white_space parameter [ , parameter ]
| GATHER white_space parameter
| GENERATE white_space ( /
| parameter
| [ parameter ] , parameter |
| [ parameter ] , [ parameter ] , parameter |
| [ parameter ] , [ parameter ] , [ parameter ] , parameter |
| INPUT white_space entity_name
| INTERN white_space expression
| LEAVE white_space parameter [ , parameter ]
| LEAVEMODEL white_space ( expression , ( integer | expression )
| ( integer | identifier ) , entity_name )
| LINK white_space parameter , ( FIFO
| LIFO
| P integer
| P expression ) [ , parameter ]
| LOGIC white_space logic_kind white_space parameter
| LOOP white_space parameter , parameter
| MARK white_space ( /
| parameter )
| MATCH white_space parameter
| MSAVEVALUE white_space parameter [ + | - ] , parameter , parameter , parameter [ , H ]
| OUTPUT white_space entity_name
| PREEMPT white_space parameter [ , PR
| , [ PR ] , parameter
| , [ PR ] , [ parameter ] , parameter
| PRINT white_space ( /
| parameter
| [ parameter ] , parameter |
| [ parameter ] , [ parameter ] , print_kind
| [ parameter ] , [ parameter ] , [ print_kind ] , expression )
| PRIORITY white_space parameter [ , BUFFER ]
| QUEUE white_space parameter [ , parameter ]
| RELEASE white_space parameter
| RETURN white_space parameter
| SAVEVALUE white_space parameter [ + | - ] , parameter [ , H ]
| SEIZE white_space parameter
| SELECT white_space ( logical_select_kind white_space parameter , parameter
| parameter [ , , , parameter ]
| min_max_select_kind white_space parameter , parameter
| relational_select_kind white_space parameter , parameter
| parameter , parameter , parameter [ , parameter ] )
| SPLIT white_space parameter , parameter [ , parameter ]
| TABULATE white_space parameter [ , parameter ]
| TERMINATE white_space ( /
| parameter )
| TEST white_space test_kind white_space parameter , parameter [ , parameter ]
| TRANSFER white_space ( BOTH , [ parameter ] , parameter
| parameter , [ parameter ] , parameter
| , parameter )
| UNLINK white_space parameter , parameter , ( parameter | ALL )
 Auxiliary rules:

white_space ::= ( \t | \r | \n | \t | \r | \n | \t | \r | \n )
new_line ::= \n
digit ::= 0-9
letter ::= a-z | A-Z | _ | $
integer ::= digit { digit }
exponent ::= ( E | e ) [ + | - ] integer
real ::= integer . [ integer ] [ exponent ]
      | [ integer ] . integer [ exponent ]
      | integer exponent
identifier ::= letter ( letter | digit )
expression ::= " anything_except_ "
value ::= [ + | - ] integer
      | [ + | - ] real
      | expression
anything_except_ ::= string of zero or more characters not containing the " character
anything_except_\n ::= string of zero or more characters not containing the new_line character
anything_except_\} ::= string of zero or more characters not containing the { character
function_kind ::= C | D | E | L | M

gate_kind ::= LS | LR | M | NM | NI | I | NU | U | SE | SF | SHE | SNF

logic_kind ::= I | R | S

print_kind ::= BLOCK | BVARIABLE
logical_select_kind ::= U
        | NU
        | I
        | NI
        | SE
        | SNE
        | SF
        | SNF
        | LR
        | LS

min_max_select_kind ::= MIN
        | MAX

relational_select_kind ::= G
        | GE
        | E
        | NE
        | LE
        | L

test_kind ::= G
        | GE
        | E
        | NE
        | LE
        | L

logical_attribute ::= FU
        | FNU
        | F1
        | FNI
        | LS
        | LR
        | SE
        | SNE
        | SF
        | SNF

relational_operator ::= "'G'"
        | "'G'"
        | "'E'"
        | "'E'"
        | "'N'"
        | "'N'"

single_sna ::= C1
        | FR
        | M1

plain_sna ::= N
        | W
        | F
        | FC
        | FR
        | FT
        | FN
        | Q
        | QA
        | QC
        | QM
        | QT
        | QK
        | QZ
matrix_sna ::= MN
   | MX
start ::= [ white_space ]
end ::= new_line
   | white_space anything_except_
new_line
_parameter ::= single_sna
   | matrix_sna ( integer | expression | $ identifier | * integer | * expression )
   | '(' ( integer | expression ) , ( integer | expression ) ')
parameter ::= value
   | identifier
   | _parameter
entity_name ::= integer
   | expression
   | identifier
function_point ::= start value , ( value | identifier | _parameter )
term ::= integer
   | real
   | expression
   | _parameter
variable_expression ::= term
   | + variable_expression
   | - variable_expression
   | variable_expression * variable_expression
   | variable_expression / variable_expression
   | variable_expression @ variable_expression
   | '(' variable_expression ')' 
b_term ::= term
   | term relational_operator term
   | logical_attribute ( integer | expression | $ identifier | * integer | * expression )
bvariable_expression ::= b_term
   | bvariable_expression + bvariable_expression
   | bvariable_expression * bvariable_expression
   | '(' bvariable_expression ')'
Chapter 7

Statement summary

In Tables 7.4, 7.5 and 7.6, a complete list of all HGPSS block declaration statements and their parameters is presented. Tables 7.7 and 7.8 list all entity declaration statements and all command statements respectively. Table 7.2 gives an overview of all Standard Numerical Attributes. There is no difference between the HGPSS Standard Numerical Attributes and those of GPSS. Contrary to the Standard Numerical Attributes, the HGPSS entity mnemonics are entirely different from those of GPSS. The HGPSS entity mnemonics are listed in Table 7.3. Table 7.1 lists and explains all metalinguistic variables used in the other tables.

The following items are relevant to a proper understanding of the tables:

- Parameters listed in adjacent columns have to be separated by commas in the program source.
- The metavariables indicating the type of the parameters are listed beneath a textual description of the parameter.
- A metavariable or literal encapsulated between [ and ] refers to an optional parameter.
- In certain cases, alternative combinations of parameters are listed top to bottom.
- Upper case letters and words are to be taken literally.
- Commas separate alternatives on the same line.
- ( . . . ) refers to a parameter list.
- All occurrences of /, ( and ) are to be taken literally.
- ± means + or −.
<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>METALINGUISTIC VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>Unsigned Integer Value</td>
</tr>
<tr>
<td>identifier</td>
<td>Identifier containing Letters, Digits, _,  and $</td>
</tr>
<tr>
<td>The First Character is a Letter, _, or $</td>
<td></td>
</tr>
<tr>
<td>expression</td>
<td>Double-quoted C++-Expression</td>
</tr>
<tr>
<td>value</td>
<td>Signed integer, Real or expression</td>
</tr>
<tr>
<td>entity_name</td>
<td>integer, expression or identifier</td>
</tr>
<tr>
<td>row</td>
<td>integer or expression</td>
</tr>
<tr>
<td>column</td>
<td>integer or expression</td>
</tr>
<tr>
<td>parameter</td>
<td>Simple SNA</td>
</tr>
<tr>
<td></td>
<td>Plain SNA followed by</td>
</tr>
<tr>
<td></td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>expression</td>
</tr>
<tr>
<td></td>
<td>$identifier</td>
</tr>
<tr>
<td></td>
<td>*integer</td>
</tr>
<tr>
<td></td>
<td>*expression</td>
</tr>
<tr>
<td></td>
<td>Matrix SNA followed by</td>
</tr>
<tr>
<td></td>
<td>integer(row,column)</td>
</tr>
<tr>
<td></td>
<td>expression(row,column)</td>
</tr>
<tr>
<td></td>
<td>$identifier(row,column)</td>
</tr>
<tr>
<td></td>
<td>*integer(row,column)</td>
</tr>
<tr>
<td></td>
<td>*expression(row,column)</td>
</tr>
</tbody>
</table>

Table 7.1: Metalinguistic variables
<table>
<thead>
<tr>
<th>SNA ENTITY</th>
<th>Single SNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Processor Value of Relative Clock</td>
</tr>
<tr>
<td>PR</td>
<td>Transaction Priority Level</td>
</tr>
<tr>
<td>M1</td>
<td>Transaction Residence Time in Model</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Plain SNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
</tr>
<tr>
<td>W</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>FC</td>
</tr>
<tr>
<td>FR</td>
</tr>
<tr>
<td>FT</td>
</tr>
<tr>
<td>FN</td>
</tr>
<tr>
<td>Q</td>
</tr>
<tr>
<td>QA</td>
</tr>
<tr>
<td>QC</td>
</tr>
<tr>
<td>QM</td>
</tr>
<tr>
<td>QT</td>
</tr>
<tr>
<td>QX</td>
</tr>
<tr>
<td>QZ</td>
</tr>
<tr>
<td>RN</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>XM</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>S</td>
</tr>
<tr>
<td>SA</td>
</tr>
<tr>
<td>SC</td>
</tr>
<tr>
<td>SR</td>
</tr>
<tr>
<td>SM</td>
</tr>
<tr>
<td>ST</td>
</tr>
<tr>
<td>TB</td>
</tr>
<tr>
<td>TC</td>
</tr>
<tr>
<td>TD</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>MP</td>
</tr>
<tr>
<td>U</td>
</tr>
<tr>
<td>UC</td>
</tr>
<tr>
<td>UH</td>
</tr>
<tr>
<td>CM</td>
</tr>
<tr>
<td>UT</td>
</tr>
<tr>
<td>BV</td>
</tr>
<tr>
<td>V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix SNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>MH</td>
</tr>
<tr>
<td>MX</td>
</tr>
</tbody>
</table>

Table 7.2: HGPSS Standard Numerical Attributes
Table 7.3: HGPSS entity mnemonics
## HGPSS Block Declaration Summary, Part 1

<table>
<thead>
<tr>
<th>BLOCK</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVANCE</td>
<td>Mean Time</td>
<td>Spread Modifier</td>
<td>[parameter]</td>
<td>[parameter]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASSEMBLE</td>
<td>Assembly Count</td>
<td>parameter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASSIGN</td>
<td>Parameter No.</td>
<td>Value to be Assigned</td>
<td>No. of Function Modifier</td>
<td>parameter</td>
<td>[parameter]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUFFER</td>
<td>Queue Name</td>
<td>No. of Units</td>
<td>parameter</td>
<td>[parameter]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEPART</td>
<td>Storage Name</td>
<td>No. of Units</td>
<td>parameter</td>
<td>[parameter]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTERMODEL</td>
<td>Model Name</td>
<td>Input Name</td>
<td>expression</td>
<td>expression</td>
<td>integer</td>
<td>expression</td>
<td>entity_name</td>
</tr>
<tr>
<td>EXTERN</td>
<td>Function Called upon Arrival</td>
<td>Function Called upon Polling</td>
<td>expression</td>
<td>expression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GATE</td>
<td>Logic Switch Name</td>
<td>Next Block if Condition is False</td>
<td>parameter</td>
<td>[parameter]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GATE</td>
<td>Name of ASSEMBLE, GATHER, or MATCH Block</td>
<td>Next Block if Condition is False</td>
<td>parameter</td>
<td>[parameter]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GATE</td>
<td>Facility Name</td>
<td>Next Block if Condition is False</td>
<td>parameter</td>
<td>[parameter]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GATE</td>
<td>Storage Name</td>
<td>Next Block if Condition is False</td>
<td>parameter</td>
<td>[parameter]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GATHER</td>
<td>Gather Count</td>
<td>parameter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GENERATE</td>
<td>Mean Time</td>
<td>Spread Modifier</td>
<td>Offset Interval</td>
<td>Limit Count</td>
<td>Priority Level</td>
<td>No. of Parameters</td>
<td>Type of Parameters</td>
</tr>
<tr>
<td></td>
<td>[parameter]</td>
<td>[parameter]</td>
<td>[parameter]</td>
<td>[parameter]</td>
<td>[parameter]</td>
<td>[parameter]</td>
<td>[F]</td>
</tr>
<tr>
<td>INPUT</td>
<td>Input Name</td>
<td>entity_name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.4: HGPSS block declaration summary, part 1

34
<table>
<thead>
<tr>
<th>BLOCK</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERN</td>
<td>Function Called upon Arrival expression</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEAVE</td>
<td>Storage Name parameter</td>
<td>No. of Units parameter [parameter]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEAVEMODEL</td>
<td>Model Name expression parameter integer identifier</td>
<td>Output Name expression integer entity_name entity_name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINK</td>
<td>Name of User Chain parameter</td>
<td>Merge Criterion parameter</td>
<td>Alternate Block expression parameter</td>
<td>FIFO expression parameter</td>
<td>FIFO expression parameter</td>
<td>LIFO expression parameter</td>
<td></td>
</tr>
<tr>
<td>LOGIC I R S</td>
<td>Logic Switch Name parameter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOOP</td>
<td>Parameter No. parameter</td>
<td>Next Block if Parameter ≠ 0 parameter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARK</td>
<td>Parameter No. parameter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATCH</td>
<td>Name of Conjugate MATCH Block parameter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSAVEVALUE</td>
<td>Matrix Name parameter</td>
<td>Row No. parameter</td>
<td>Column No. parameter</td>
<td>Value to be Saved parameter</td>
<td>Matrix Type parameter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Output name entity_name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PREEMPT</td>
<td>Facility Name parameter</td>
<td>Priority Option parameter</td>
<td>Next Block for Preempted Transaction parameter [PR]</td>
<td>Parameter No. of Preempted Transaction parameter</td>
<td>Remove Option parameter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINT</td>
<td>Lower Limit parameter</td>
<td>Upper Limit parameter</td>
<td>Entity Name parameter</td>
<td>Mnemonic parameter</td>
<td>File Name parameter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIORITY</td>
<td>New PR Value parameter</td>
<td>Buffer Option parameter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUEUE</td>
<td>Queue Name parameter</td>
<td>No. of Units parameter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RELEASE</td>
<td>Facility Name parameter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETURN</td>
<td>Facility Name parameter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAVEVALUE</td>
<td>Savevalue Name parameter</td>
<td>Value to be Saved parameter</td>
<td>Savevalue Name parameter</td>
<td></td>
<td>Type parameter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEIZE</td>
<td>Facility Name parameter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.5: HGPSS block declaration summary, part 2
<table>
<thead>
<tr>
<th>BLOCK</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>$U,NU$</td>
<td>$I,NI$</td>
<td>$SE, SNE$</td>
<td>$SF, SNF$</td>
<td>$LR, LS$</td>
<td>Parameter in which to Place Entity No.</td>
<td>Lower</td>
</tr>
<tr>
<td>SELECT</td>
<td>$MIN$</td>
<td>$MAX$</td>
<td>Parameter in which to Place Entity No.</td>
<td>Lower</td>
<td>Upper</td>
<td>Limit</td>
<td>Limit</td>
</tr>
<tr>
<td>SELECT</td>
<td>$G$</td>
<td>$GE$</td>
<td>$E$</td>
<td>$NE$</td>
<td>$LE$</td>
<td>Parameter in which to Place Entity No.</td>
<td>Lower</td>
</tr>
<tr>
<td>SPLIT</td>
<td>No. of Offspring</td>
<td>Next Block for Offspring</td>
<td>Serialization Parameter</td>
<td>No. of Parameters for each Offspring</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABULATE</td>
<td>Table Name</td>
<td>Weighting Factor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERMINATE</td>
<td>Termination Counter Decrement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST</td>
<td>$G$</td>
<td>$GE$</td>
<td>$E$</td>
<td>$NE$</td>
<td>$LE$</td>
<td>First Value</td>
<td>Second Value</td>
</tr>
<tr>
<td>TRANSFER</td>
<td>Selection Mode</td>
<td>First Block Examined</td>
<td>Second Block Examined</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSFER</td>
<td>Selection Mode</td>
<td>First Block Examined</td>
<td>Second Block Examined</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSFER</td>
<td>Selection Mode</td>
<td>Next Block Entered</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNLINK</td>
<td>Name of User Chain</td>
<td>Next Block for Unlinked Transactions</td>
<td>Unlink Count</td>
<td>Parameter No.</td>
<td>Match Argument</td>
<td>Alternate</td>
<td>Exit</td>
</tr>
</tbody>
</table>

Table 7.6: HGPSS block declaration summary, part 3
<table>
<thead>
<tr>
<th>ENTITY</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BVARIABLE</strong></td>
<td>Combination of Numeric Data Specifications, Relational Operators, Logical Attributes and Boolean Operators</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Numeric Data Specifications</td>
<td>Relational Operators</td>
<td>Logical Attributes</td>
<td>Boolean Operators</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>'G', 'GE'</td>
<td>'L', 'LE'</td>
<td>'E', 'NE'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FU, FNU</td>
<td>FI, FNI</td>
<td>SE, SNE</td>
<td>SF, SNF</td>
<td></td>
</tr>
<tr>
<td><strong>BVARIABLE</strong></td>
<td>Function Name</td>
<td>/expression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function Name</td>
<td>Function Type and No. of Points</td>
<td>Data points separated by / or - new line</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Cinteger</td>
<td>value,value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Dinteger</td>
<td>value,identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Einteger</td>
<td>value,parameter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Linteger</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FUNCTION</strong></td>
<td>Function Name</td>
<td>/expression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function Name</td>
<td>Argument</td>
<td>Function Type</td>
<td>and No. of Points</td>
<td>Data points separated by / or - new line</td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Cinteger</td>
<td>value,value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Dinteger</td>
<td>value,identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Einteger</td>
<td>value,parameter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Linteger</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FUNCTION</strong></td>
<td>Function Name</td>
<td>/expression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function Name</td>
<td>Argument</td>
<td>Function Type</td>
<td>and No. of Points</td>
<td>Data points separated by / or - new line</td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Cinteger</td>
<td>value,value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Dinteger</td>
<td>value,identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Einteger</td>
<td>value,parameter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Linteger</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MATRIX</strong></td>
<td>Matrix Type</td>
<td>Number of Rows</td>
<td>Number of Columns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X, H</td>
<td>integer, expression</td>
<td>integer, expression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SUBMODEL</strong></td>
<td>Submodel Type and Arguments</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>identifier[...]]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>QTABLE</strong></td>
<td>Name of Queue</td>
<td>Inclusive Upper Limit of Lowest Frequency Class</td>
<td>Width of Intermediate Frequency Classes</td>
<td>Number of Frequency Classes</td>
<td></td>
</tr>
<tr>
<td>entity,name</td>
<td>value</td>
<td>value</td>
<td>value</td>
<td>integer, expression</td>
<td></td>
</tr>
<tr>
<td><strong>STORAGE</strong></td>
<td>Storage Capacity</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>integer, expression</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TABLE</strong></td>
<td>Table Arguments</td>
<td>Inclusive Upper Limit of Lowest Frequency Class</td>
<td>Width of Intermediate Frequency Classes</td>
<td>Number of Frequency Classes</td>
<td>Time Interval for RT-Table</td>
</tr>
<tr>
<td>parameter, IA, RT</td>
<td>value</td>
<td>value</td>
<td>[W]integer</td>
<td>[value]</td>
<td>[W]expression</td>
</tr>
<tr>
<td>parameter, IA, RT</td>
<td>value</td>
<td>value</td>
<td>[W]expression</td>
<td>[value]</td>
<td></td>
</tr>
<tr>
<td><strong>VARIABLE</strong></td>
<td>Combination of Numeric Data Specifications and Arithmetic Operators</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Numeric Data Specifications</td>
<td>Arithmetic Operators</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>+, -, /, *, @</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>VARIABLE</strong></td>
<td>Function Name</td>
<td>/expression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function Name</td>
<td>Argument</td>
<td>Function Type</td>
<td>and No. of Points</td>
<td>Data points separated by / or - new line</td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Cinteger</td>
<td>value,value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Dinteger</td>
<td>value,identifier</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Einteger</td>
<td>value,parameter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Linteger</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.7: HGPSS entity declaration summary
<table>
<thead>
<tr>
<th>COMMAND</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOWN</td>
<td>Submodel</td>
<td>Name entity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>END</td>
<td>Logic Switch</td>
<td>to be Set</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>L$integer$</td>
<td>L$expression$</td>
<td>L$identifier$</td>
<td></td>
</tr>
<tr>
<td>INITIAL</td>
<td>Matrix</td>
<td>Initial</td>
<td>MX$integer$(row, column)</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>Savevalue</td>
<td>Value</td>
<td>MX$expression$(row, column)</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MH$integer$(row, column)</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MH$expression$(row, column)</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MHS$identifier$(row, column)</td>
<td>value</td>
</tr>
<tr>
<td>INITIAL</td>
<td>Savevalue</td>
<td>Initial</td>
<td>X$integer$</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X$expression$</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>XS$identifier$</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>XH$integer$</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>XH$expression$</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>XHS$identifier$</td>
<td>value</td>
</tr>
<tr>
<td>JOB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINT</td>
<td>Lower Limit</td>
<td>Upper Limit</td>
<td>Entity Name</td>
<td>File [expression]</td>
</tr>
<tr>
<td></td>
<td>[parameter]</td>
<td>[parameter]</td>
<td>[mnemonic]</td>
<td>[expression]</td>
</tr>
<tr>
<td>RESET</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMULATE</td>
<td>Model Type</td>
<td>File Name</td>
<td>and Arguments</td>
<td>[expression]</td>
</tr>
<tr>
<td>START</td>
<td>Initial Counter Value</td>
<td>Termination</td>
<td>integer, expression</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.8: HGPSS command summary
<table>
<thead>
<tr>
<th>Bibliography</th>
</tr>
</thead>
</table>
| [Boehm & Jacopini 1966] | Boehm, Corrado  
Jacopini, Giuseppe  
*Flow diagrams, Turing machines, and languages with only two formation rules*  
Communications of the ACM, vol. 9, no. 5  
1966 |
| [Claeys 1992] | Claeys, Filip  
*HGPSS: Object-georiënteerde “process-interaction” simulatie*  
M.Sc. in Computer Science thesis  
Universiteit Gent  
Ghent, Belgium 1992 |
*System Simulation*  
Prentice-Hall  
Brown, Doug  
*LEX & YACC*  
O’Reilly & Associates, Inc.  
Sebastopol, California, U.S.A. 1990 |
| [Neelamkavil 1987] | Neelamkavil, Francis  
*Computer simulation and modelling*  
John Wiley & Sons  
1987 |
| [Schriber 1974] | Schriber, Thomas J.  
*Simulation Using GPSS*  
John Wiley & Sons  
New York, New Jersey, U.S.A. 1974 |
| [Stroustrup 1987] | Stroustrup, Bjarne  
*The C++ Programming Language*  
Addison-Wesley  
1987 |
| [Zeigler 1976] | Zeigler, B.P.  
*Theory of Modelling and Simulation*  
John Wiley & Sons  
New York, New Jersey, U.S.A 1976 |