Proceedings of the 2009 Winter Simulation Conference
M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, eds.

JGPSS, AN OPEN SOURCE GPSS FRAMEWORK TO TEACH SIMULATION

Pau Fonseca i Casas
Josep Casanovas

Dept. of Statistics and Operational research
Universitat Politècnica de Catalunya
08034, Barcelona, SPAIN

ABSTRACT

GPSS has been used for years to teach simulation. Different tools following the GPSS syntax exist. Usually these tools can be used to construct simulation models helping in the teaching of simulation. However no available framework is capable to simplify the development of a complete simulation tool following the GPSS syntax. This paper presents JGPSS, a framework based in Java language that can be used by students to build a complete simulation tool following the GPSS syntax. JGPSS has two versions, one capable to perform simulations and obtain statistical information, and other designed as a framework for students of computer sciences. This last framework, described in this paper, lacks in the implementation of the main simulation engine algorithms or the related structures. This framework allows the computer science student to reach a deeper understanding of how to construct a complete process interaction simulation engine, the paradigm that GPSS follows.

1 INTRODUCTION

In the origin GPSS means *Gordon's Programmable Simulation System*, in honor to Geoffrey Gordon, its creator. However, when the system started to grow he decided to change the name to *General Purpose Simulation System* (Ståhl 2001). In 1961 IBM released the first version of the GPSS system, and later other different versions like GPSS/360 and GPSS V (Gordon, 1978). Although more than 40 years have passed since this first release, an intense development effort is still being done, and a lot of different versions exist, like GPPS/H, GPSS World or WebGPSS.

GPSS/H (Schriber 1991; Banks, et al. 1995; Crain and Henriksen 1999; Crain 1997) is distributed by Wolverine Software Corporation, <http://www.wolverinesoftware.com/>, founded on 1976 by James O. Henriksen. A lot of books describes the particularities of this GPSS version. Some of them can be found in Wolverine website. This version of GPSS does not include a graphical interface (runs on DOS); hence the models can be defined using a conventional text editor. However, tools to allow a graphical representation of the models are distributed by the same company, like Proof Animation tool.

Minuteman GPSSWorld <http://www.minutemansoftware.com/> (Minuteman Corp.), presents a graphical interface with an embedded text editor that allows the definition of the model inside the tool itself. The user can also find in a window all the defined GPSS blocks implemented in the tool, simplifying the modeling process. GPSSWorld presents some other interesting features, like the capability to represent the movement of the transactions over the different elements of the model. Additionally a powerful language allows a complete definition of the more complex behavior (the Plus syntax). A complete online manual can be found in the website of minuteman.

WebGPSS system <http://www.webgpss.com/> (Born and Ståhl 2004), is able to specify the definition of the model in a graphical way. Each one of the different GPSS blocks has a graphical representation that can be used to build the models through a flow diagram. Although this representation is not a specification by itself, it does simplify the understanding of the model behavior for the non experienced GPSS users. The last version is not currently working in Java, but is planned a new release for 2008 adding the new capabilities of the new Sun JDK.

Each one of these GPSS versions allows the student to understand how the GPSS system works and they are excellent ways to introduce the student in the world of the discrete simulation following the process interaction paradigm. However in order to deeply understand how the simulation engine works, knowledge needed for a computer science student, the best exercise is to construct one by itself. JGPSS, additionally to help in the construction of a GPSS simulation model, includes, as main objective, to become an infrastructure guiding the student in the process of constructing an interaction process paradigm based simulation tool.

2 PROCESS INTERACTION PARADIGM

Discrete simulation has three main discrete paradigms, (i) activity scanning, (ii) event scheduling and (iii) process interaction (Law and Kelton 2000). Each one of these paradigms has its benefits and problems, and they are usually explained in a introductory simulation course.

GPSS was one of the first systems to introduce the process interaction paradigm (Ståhl 2001). In this formalism a transaction is moving through the blocks diagrams representing the system. This flow of the transactions over the different blocks can cause a competition for resources with other transactions concurrently traveling through the system. This can cause delays in the normal transaction movement.

In a process interaction paradigm two kinds of delays can be represented, with two different hypothetical processes, formally named P1 and P2. P1 processes are those who require a service for a finite time. Since the amounts of resources are limited, and other P1 resources can need the same service, conditional blocking situations can be produced. The second processes, P2, represent those who implement a delay in its normal evolution. For instance the time needed for an entity to reach a specific position in the space. In these processes the delay is not caused by the competition for resources. Of course that in a real process P1 and P2 processes behavior is represented mixed.

In the next figure we show a chronogram showing the interaction between P1 and P2 processes.

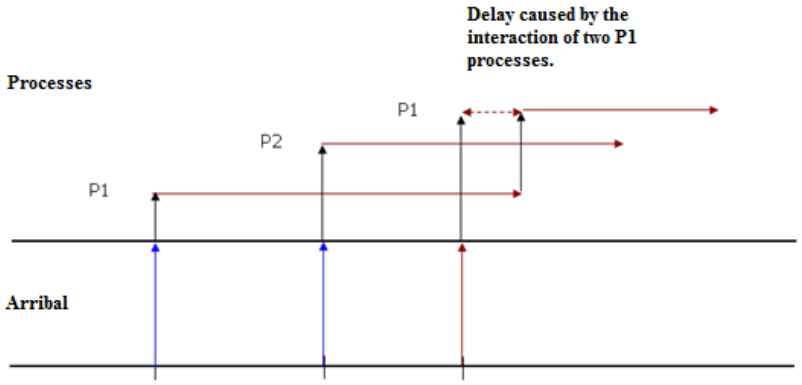


Figure 1: Interaction process chronogram

2.1 The Entity List

Since interaction process is a discrete paradigm, the modifications on the model state variables are defined by some events that occur depending on different conditions. In an event scheduling paradigm the structure defining these events is the "event list". In a interaction process paradigm, since all the events are related with an entity (that is performing an action of its process) we can define an "entity list".

The entity is a structure defined as follows:

```
Struct entity
{
    MoveTime Real;
    CreationTime Real;
    Priority integer;
    CurrentProcess Process;
    CurrentOperation Operation;
}
```

MoveTime defines when the event can be processed; CreationTime defines when the event has been created. Since the event is related with an entity, this allows obtaining the live time statistic. CurrentProcess defines the process that the entity is executing, and CurrentOperation defines the operation the entity wants to execute.

Two main types of entities exist: the NOW entities and the NEXT entities. NOW entities have a MoveTime equal or less than the simulation clock and the NEXT entities have a MoveTime bigger than the simulation clock (representing the future events). Usually these entities are stored in two different lists, the NOW list is sorted by priority, while the NEXT list is sorted by MoveTime.

2.2 The Simulation Loop

The next figure summarizes how an interaction process simulation loop works.

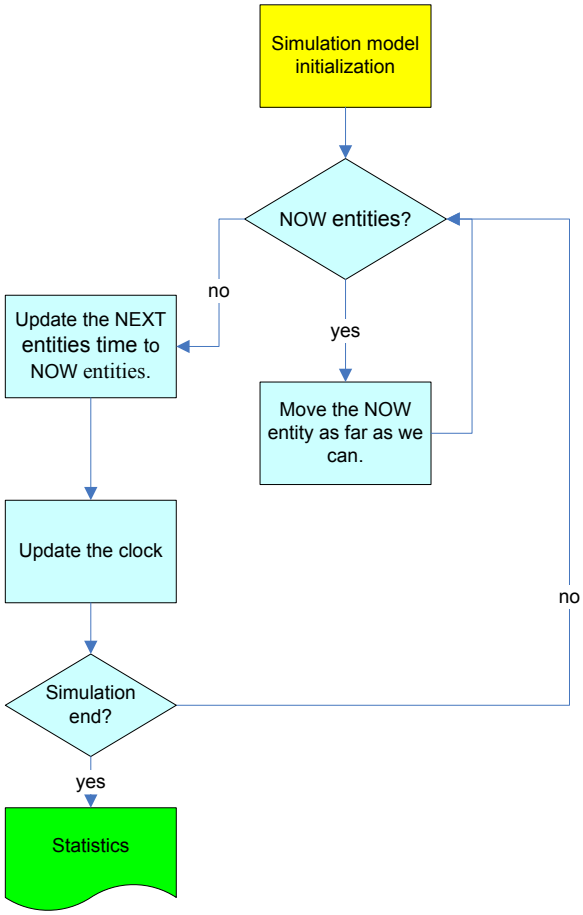


Figure 2: Interaction process main loop

The main idea is to perform all the actions we can with all the NOW entities before we need to update the clock.

First we take all the NOW entities from the event list one by one (sorted by priority). For each one of the entities we try to execute its related actions. Three alternatives can be done. First, since the entity can be in a P2 process, a delay can cause that the entity must queue to the NEXT list, since the delay implies that this entity cannot continue its movement now. Second, the entity can be related to a P1 process, and although no delay exists, the entity cannot continue its movement since an element is blocking its path. The third case is when nothing blocks the entity movement and it leaves the system. When all the NOW entities have been processed we need to modify the simulation clock value with the time of the MoveTime of the first NEXT entities. Now we can start again the loop.

2.3 Why GPSS

The main idea that guided the development of GPSS was the creation of a tool that can be used by anyone, although could not have programming skills. It was developed as a simulation tool for non programmers; therefore the capability of graphical representation of the model can be a very interesting feature.

Other tools exist to represent a model following the process interaction paradigm. However, these tools although powerful, present a set of nodes with a complex behavior in comparison with the GPSS blocks. That means that, keeping in mind that the objective is that the student could be capable to implement the tool, GPSS is a good alternative, since it permits a relative simple implementation for each one of its different operations (blocks). In the past, the students develop from scratch a simulation engine that must be capable of represent a model (selected by the students). This code usually was based on Java or C++ languages. The students usually spend too much time developing classes and code related with the common parts of a computer program, the GUI or the disc management structures (code not specifically related with the simulation discipline).

Now with JGPSS the student accelerates the development of the simulation engine using Java, focusing on the development of the simulation classes. Using SEEQ (Student Experience of Education Questionnaire) questionnaires we detect that since from the beginning the student see results, they are more motivated.

3 GPSS LANGUAGE

As we said, the GPSS language follows the process interaction paradigm. That means that the model definition has at least three elements, entities, process and operations belonging to the process.

In GPSS the operations are named blocks, and the entities are named transactions. Each one of the blocks performs some operations with the entities, allowing the representation of the model behavior evolution. In GPSS the NOW activities (NOW transactions) are stored in the CEC queue (Current Event Chain), while the NEXT transactions are stored in the FEC queue (Future Event Chain). While the CEC sorts its elements by priority, the FEC sorts its elements by time and priority. In the next figures we are showing a hypothetical CEC and FEC and the GPSS algorithm.

xact id	curBlk	nxtBlk	moveTime	priorityLevel
5	7	8	...	20
3	12	13	...	16
8	9	10	...	12

Figure 3: GPSS CEC

xact id	curBlk	nxtBlk	moveTime	priorityLevel
7	3	4	42.6	3
9	Neix	19	47.6	15
2	7	8	51.9	12
11	32	33	51.9	16

Figure 4:GPSS FEC

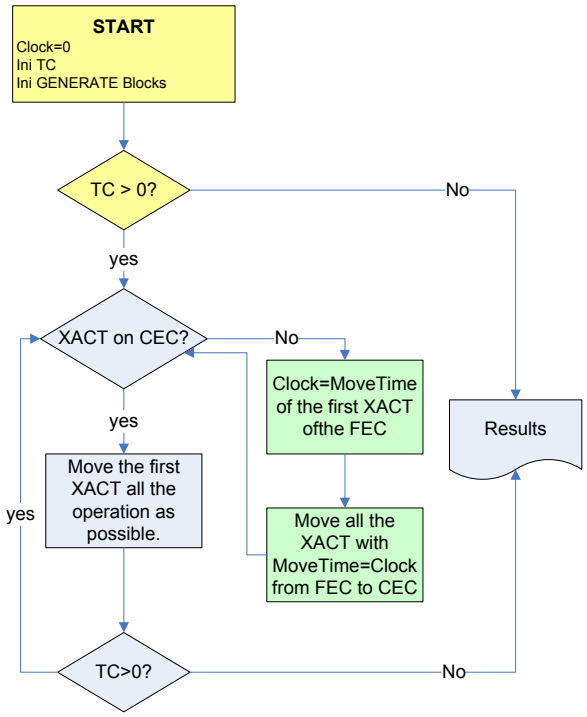


Figure 5: GPSS main loop algorithm

The TC (transactions counter) defines the termination condition of the simulation. Is an integer that is decremented with the value represented in the TERMINATE block parameter, and is initialized with the value of the START statement.

3.1 GPSS Blocks.

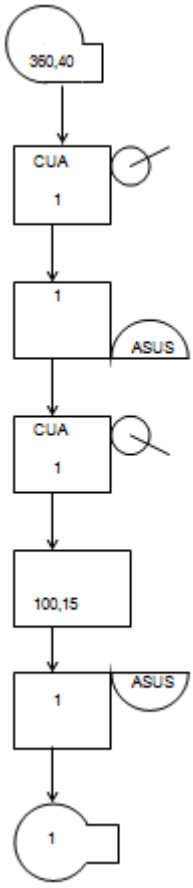
Since all the blocks have a graphical representation, the definition of a GPSS process can be represented graphically. In the Figure 6 we show a simple process representing a server with a queue in textual and graphical representation. The name of the queue is CUA, while the name for the resource is ASUS. The code follows the GPSSWorld syntax.

The number of the blocks implemented in GPSS depends on the version. In the Table 1 the blocks implemented in the three versions analyzed in the introduction of this paper can be reviewed. For a comparison of other versions of GPSS depending on the number of blocks implemented (Ståhl 2007) can be reviewed.

Table 1: GPSS blocks of different versions.

	gpss/h	world	webgpss	jgpss
adopt		x		
advance	X	x	x	x
alter	X	x		
atnwait	X			
arrive			x	
assemble	X	x	x	x
assign	X	x		
branch				
buffer	X	x		x
bclose	X			
bfiledef	X			
bgetlist	X			
bgetstring	X			
blet	X			
bputpic	X			
bputstring	X			
bclear	X			
bcall	X			
breset	X			
change	X			
count	X			
close		x		
compare				
count		x		
depart	X	x	x	x
displace		x		
enter	X	x	x	x
examine	X	x		
execute	X	x		
favail	X	x		x
funavail	X	x		x
gate	X	x		x
gather	X	x		x
generate	X	x	x	x
goto			x	
help	X		x	
if			x	
index	X	x		

	gpss/h	world	webgpss	jgpss
integration		x		
join	x	x		
leave	x	x	x	x
let			x	
link	x	x		
logic	x	x		x
loop	x	x		x
mark	x	x		
match	x	x		x
msavevalue	x	x		x
open		x		
plus		x		
preempt	x	x		
print	x		x	
priority	x	x		x
queue	x	x		x
read		x		
release	x	x	x	x
remove	x	x		
reschedule	x			
return	x	x		
savail	x	x		x
savevalue	x	x		x
scan	x	x		
seek		x		
seize	x	x	x	x
select	x	x		
split	x	x	x	x
sunavail	x	x		x
tabulate	x	x	x	
terminate	x	x	x	x
test	x	x		x
trace	x	x		
transfer	x	x		x
unlink	x	x		
untrace	x	x		
waitif			x	
write		x		




```
ASUS    STORAGE     2

    SIMULATE

    GENERATE    360,40
    QUEUE       CUA,1
    ENTER       ASUS,1
    DEPART      CUA,1
    ADVANCE     100,15
    LEAVE       ASUS,1
    TERMINATE   1
```

Figure 6: Simple GPSS process

4 JGPSS ARCHITECTURE

Since the infrastructure is programmed in Java, it is formed by a set of classes. The dependencies and the relation of the different classes are represented in the Figure 7. The students focus their developments in the "Simulation Engine" classes, defining the process interaction main simulation loop. These classes need the structures defined in the "Model Structures" set of classes and are filled with the graphic interface by the user, or recovering a text file that defines a model, using the "File manager" set of classes.

In brief, the students have, thanks JGPSS, the implementation of a framework capable of save, represent and recover a GPSS model. Also JGPSS have templates for the main structures needed to implement a process interaction simulation engine. JGPSS acts as a framework that guides the student in the implementation of the needed structures that defines a simulation engine. The students work is related with the implementation of the simulation engine, the events management (related with the blocs behavior), the statistics acquisition (SNA's Standard Numeric Attributes) and the experimentation framework. This implies that the final result will be better and more similar to a complete product, helping in the motivation of the students.

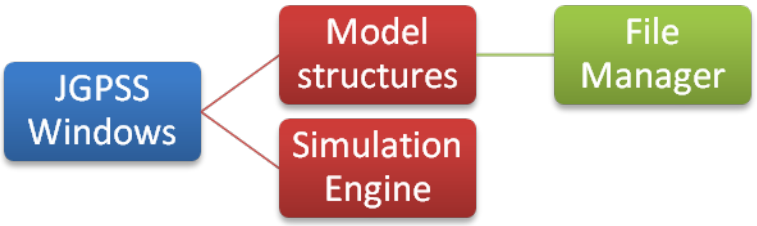


Figure 7: JGPSS architecture

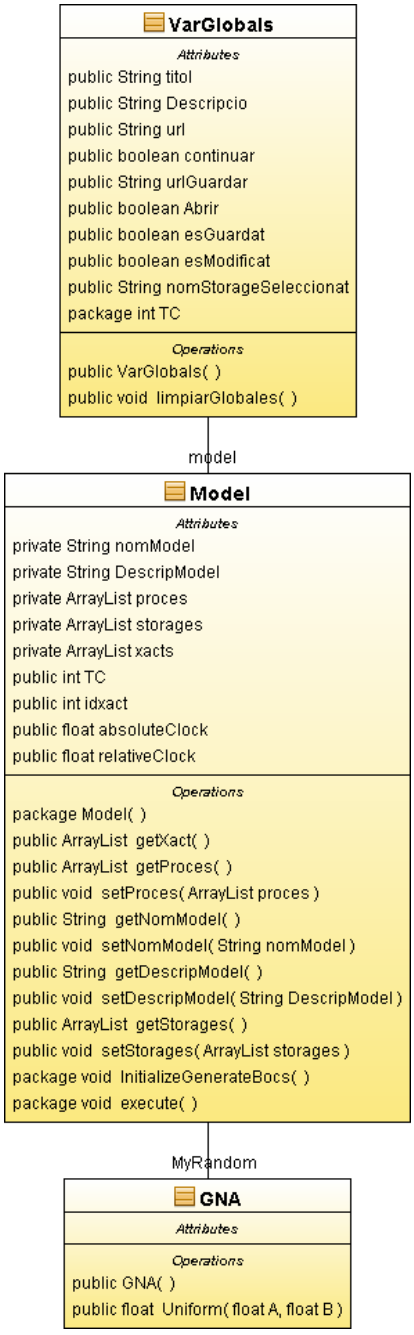


Figure 8: Simulation engine main classes

In the Figure 8 the classes that implement the simulation loop are shown. Basically the main loop is contained in the "execute()" method of "Model" class. The student also must implement the needed RNG (Random Number Generator) and the probability distributions for a specific problem. As an example for the students, the implementation of a Uniform distribution using random() java method is coded. In red are the set of classes that the students usually modify.

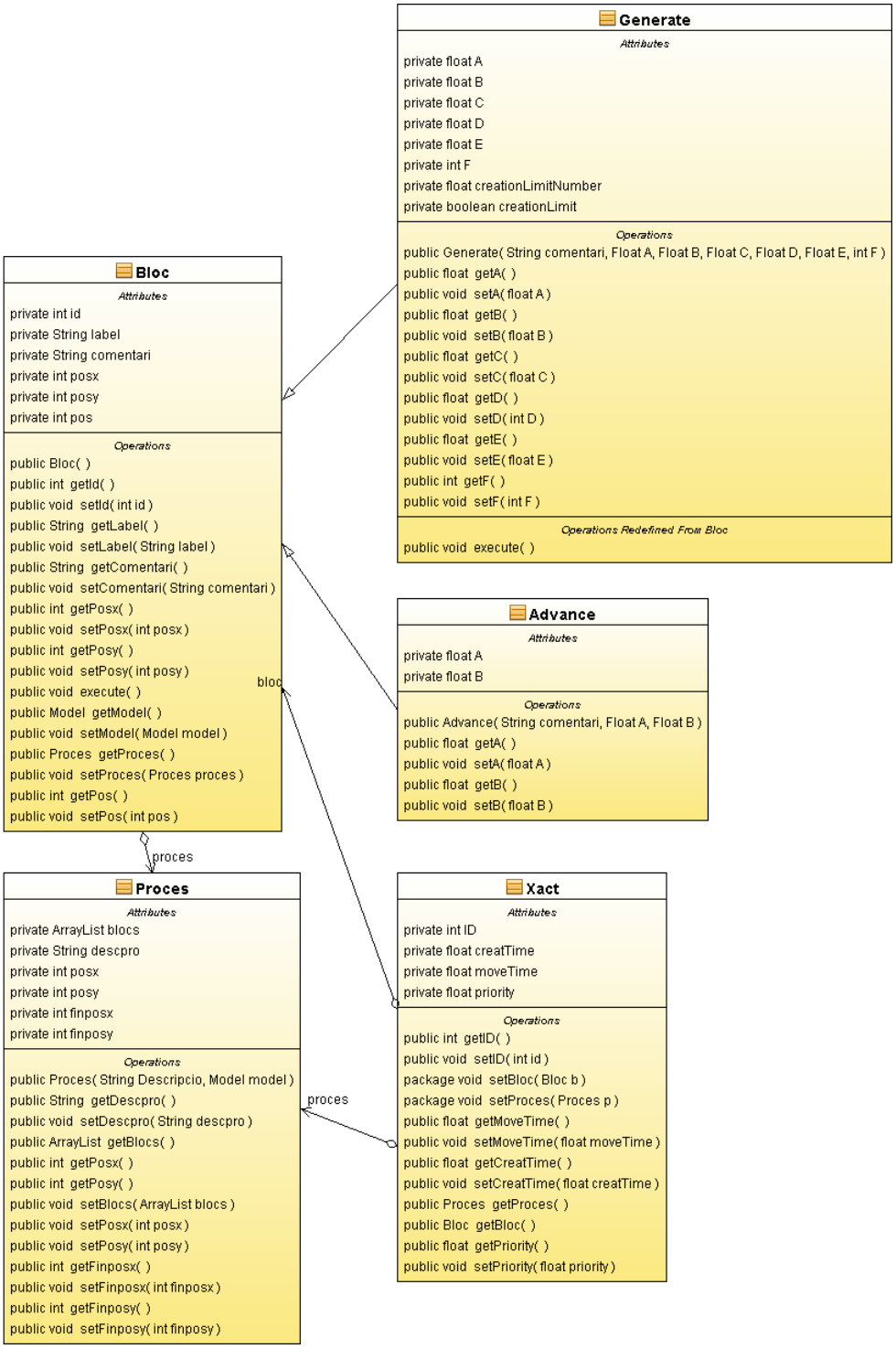


Figure 9: Model structure classes

"Model structures" classes allows to define the process and the blocks of the model. Also each one of the different blocks are implemented using a class, for instance we have a class for the GENERATE block, a class for the ADVANCE block, and so on. Each one of the blocks have a method named execute() that allows the definition of its behavior. In section 5.2 an example of execute() method for the GENERATE block is shown.

5 JGPSS MODEL FILE

The JGPSS model is stored in a simple text file as the represented below.

```
*
* Demo
* Demo Model
*

*
* main
*
     Generate        1.0,0.5,0.0,0.0,0.0,0     ;New entities
     Queue           Q,0     ;Entering in the queue Q
     Seize           MAQ     ;Entering in the server MAQ
     Depart          Q,0     ;Leaving the queue Q
     Advance         1.0,0.0     ;Service time
     Release         MAQ     ;Leaving in the server MAQ
     Terminate       0     ;Leaving the system.

*
* Clock
*
     Generate        1440.0,0.0,0.0,0.0,0.0,0     ;Generating the time.
     Terminate       1     ;Clock process terminate
```

The user can define the models writing the processes with a common text editor. As we see in the example, each one of the different processes starts with a comment containing the process name. However an existing GUI simplifies the work creating the model without the need of writing a single line of GPSS code.

5.1 JGPSS GUI

Based on Swing classes the JGPSS interface describes the GPSS process in a graphical way. The advantages or disadvantages related to the use of a graphical interface to construct a simulation model are discussed in (Crain 1997). JGPSS have a graphical interface to motivate the students to build a complete process interaction simulator. Also, since the models are simple text files, the specialized user can write the code directly without the need of the GUI. However, focusing on how the JGPSS GUI works, the user must first define the name and the description of the model (in a modeless window). Next the user can define the different processes of the model with the button located on the bottom right corner. If the user clicks with the mouse right button in each one of the different blocks a contextual menu containing different options is shown. In Figure 10 some windows of the GUI are shown (the definition of a simple GPSS model containing two process, the *Create process* button and the contextual menu).

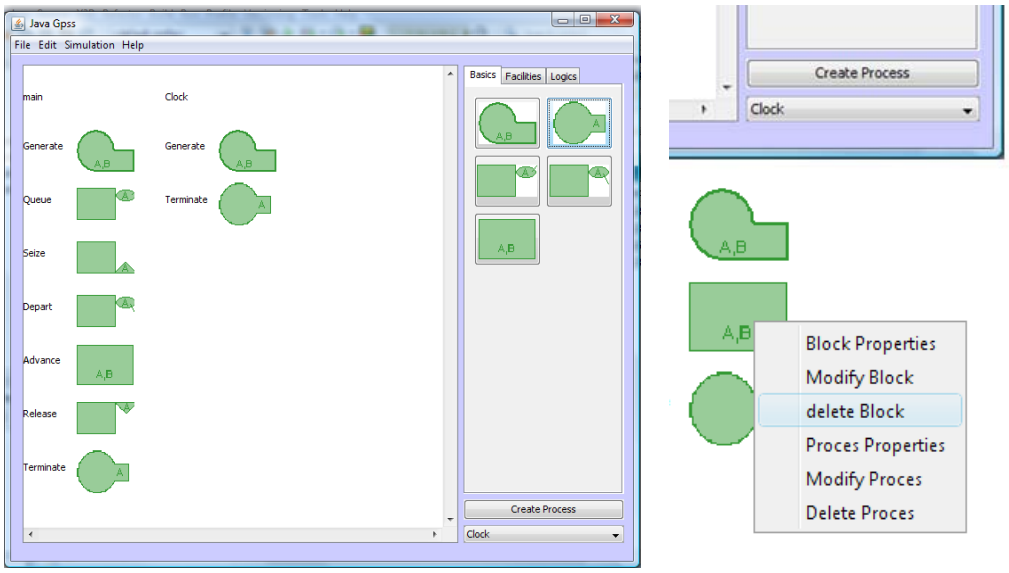


Figure 10: JGPSS main window, *Create process* button and contextual menu

The main objective of this GUI is to store the structure of the model, the processes and the blocks that compose each one of the processes, when the user defies a new model, or when the user opens a file containing a model (from a file with the format we see in previously). The student focuses his effort in the definition of the simulation engine code with no need to build the simulation structures by itself (or a graphical interface to load these structures).

5.2 JGPSS Codification Guide for Students

As the main objective of this framework is that the student directs his attention to program the simulation engine, some areas to insert the code appear. As an example, the code shows the area where the student must write the simulation engine loop.

```
void execute() {
    Xact xact;
    //Initialization
    relativeClock=0;
    absoluteClock=0;
    InitializeGenerateBocs();

    //Simulation engine loop
    while (TC > 0) {
        //TODO 1: First XACT.
        //TODO 2: Move the XACT as far as we can.
        //TODO 3: Look for other NOW XACT.
        //TODO 4: CLOCK UPDATE PHASE
        //TODO 5: Move the Xacts of the FEC to the CEC.
        //TODO 6: Goto TODO 1.
    }
}
```

The student work is structured in different steps. First defining the needed structures to store the transactions. By default the transactions are stored in a single ArrayList, but the student must decide if this structure is enough (for instance if two structures are needed, one for the CEC and other for the FEC), or in other sense, if this structure must be modified by another more efficient in order to obtain good performance accessing to the items that keep sorted. Once this structure is defined and implemented, the simulation loop must be implemented. This loop must follow the algorithms shown in the Figure 5. The main work consists in the implementation of the behavior for each one of the different JGPSS blocks. Each of these blocks have a method "execute()" that must be rewritten in order to define this behavior. As an example we discuss the code needed for execute the GENERATE bloc.

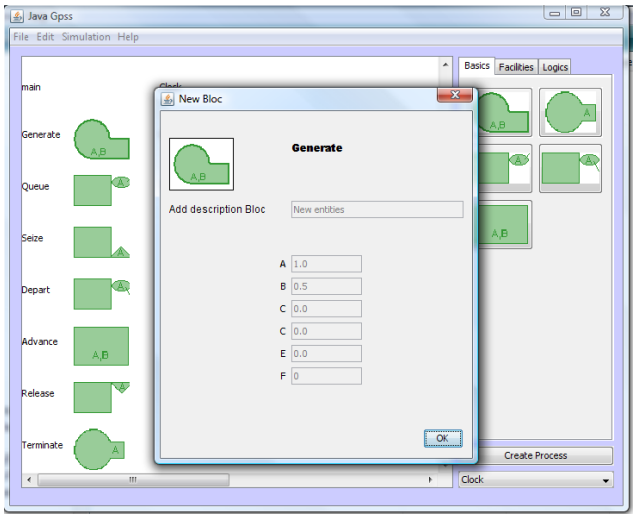


Figure 11: GENERATE block parameters

The parameters for a GENERATE block in a JGPSS model are (A) Intergeneration time ,(B) Half range or Function Modifier, (C) Start delay time, (D) Creation limit, (E) Priority, and (F) to define if the parameter C must be used. These parameters can be defined graphically as we can see in the Figure 11. As an example, the code that represents the behavior of the GENERATE block is shown in the next code. Of course, the student is free to rewrite this code.

```
@Override
public Block execute(Xact Tr) {
    if ((creationLimit && creationLimitNumber > 0) || !creationLimit) {
        Xact xact;
        xact = new Xact();
        xact.setBloc(this);
        xact.setProces(getProces());
        xact.setCreatTime(getModel().relativeClock);
        xact.setPriority(E);
        //F parameter determines if the C parameter is valid (improve it!)
        if (getModel().relativeClock == 0 && F > 0) {
            xact.setMoveTime(C);
        } else {
            xact.setMoveTime(getModel().MyRandom.Uniform(A, B));
        }
        getModel().idxact++;
        xact.setID(getModel().idxact);
        getModel().getXact().add(xact);
        creationLimitNumber--;
        System.out.println("Generate DONE.");
    } else {
        System.out.println("Creation limit reached.");
    }
    return (Tr.nextBlock());
}
```

This block is presented as a template that the student can use to implement all the other blocks. However, it depends on the structures the student use to define the lists and all the other elements.

6 CONCLUDING REMARKS

In this paper we introduce a first approach to JGPSS software that is presented in two different flavors: as a simulation tool that allows the execution of a simulation model following the GPSS language, or as an infrastructure that helps a computer science student to implement an interaction process simulation engine following the structure of the well known simulation language GPSS. Since the infrastructure stores all the information related to the model and presents a clear graphical interface, the student focuses his efforts in the implementation of the pieces of code related to the simulation engine and its struc-

tures. Also, since the final result is a graphical and quite complete application, the student is more motivated and receptive to understand how the engine works, in order to see a graphical final result.

JGPSS can be a valuable tool for computer science students to deeply understand how an interaction process simulation engine works, and a valuable tool for teachers in order to guide the programming skills of the students, focusing in the more complex areas of the simulation engine. This tool also improves the motivation of the students. Some of the comments received using a SEEQ (Student Experience of Education Questionnaire) tests shows that since the final result of the work is quite similar to a complete product, motivates the student to work harder than usual, and also increases the implication with the course.

In the next future we are planning to release both versions of JGPSS for teaching purposes in other universities and interested institutions, in order to extend the experience of our computer science students with this new flavor of a conceptually excellent and well known tool: the GPSS language.

ACKNOWLEDGMENTS

Special thanks to Maria Dolores Alvarez Arpal for her contribution in the development of the initial version of the system.

REFERENCES

Banks, J, J. Carson, and J. Sy. 1995. *Getting Started With GPSS/H*. Wolverine Software Corporation.

Born, R.G., and I. Ståhl, 2004. WEBGPSS: the first two hours of simulation education, In *Proceedings of the 2004 Winter Simulation Conference*, eds. M. D. Rossetti, J. S. Smith, and B. A. Peters R .G. Ingalls, 2066-2074. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Crain, R. C., and J. O. Henriksen, 1999. Simulation using GPSS/H, In *Proceedings of the 1999 Winter Simulation Conference*, eds. P. A. Farrington, et al., 182-187. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Crain, R. 1997. Simulation using GPSS/H, In *Proceedings of the 1997 Winter Simulation Conference*, eds. K. J. Healy, D. H. Withers, B. L. Nelson, and S. Andradóttir, 567-573. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Gordon, G. 1978. *System simulation*. Second edition. New Yersey : Prentice-Hall. Inc.

Law, A. M., and D.W. Kelton, 2000. *Simulation Modeling and Analysis*, 3rd ed. New York: McGraw-Hill, Inc.

Lorenz, P. GPSS/H Blocks. University of Magdeburg. Available via <http://isgwww.cs.uni-magdeburg.de/pelo/sa/sim1.php> [accessed: May 3, 2009.]

Minuteman Corp. GPSSWorld manual. Available via <http://www.minutemansoftware.com/tutorial/tutorial_manual.htm> [accessed: May 3, 2009.]

Schriber, T. J. 1991. *An Introduction to Simulation Using GPSS/H*. New York : John Wiley & Sons.

Schriber, T. J. 1988. Perspectives on simulation using GPSS, In *Proceedings of the 1988 Winter Simulation Conference*, eds. M. Abrams, P. Haigh and J. Comfort, 71-84. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Ståhl, I. 2001. GPSS – 40 years of development, In *Proceedings of the 2001 Winter Simulation Conference*, eds. B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, 577-585. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Ståhl, I. 2007. Teaching simulation to business students summary of 30 years' experience, In *Proceedings of the 2007 Winter Simulation Conference*, eds. B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R.Barton S. G. Henderson. 2327-2336. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

PAU FONSECA I CASAS is a Professor of the Department of Statistics and Operational research of the Technical University of Catalonia. He obtained his degree in computer engineering on 1999 and his Ph.D. on 2007 from Technical University of Catalonia. He works in the LCFIB (Barcelona informatics school laboratory) developing Simulation projects since 1998, also is member of LogiSim, group dedicated to the research and simulation tools and projects development. His email is <pau@fib.upc.edu>.

JOSEP CASANOVAS is a Professor of the Department of Statistics and Operational research of the Technical University of Catalonia. He is the director of the LCFIB (Barcelona informatics school laboratory) and director of LogiSim, group dedicated to the research and simulation tools and projects development. His email is <josepk@fib.upc.edu>.