Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

PASION TUTORIAL

Stanislaw Raczynski

Universidad Panamericana
Augusto Rodin 498
03910 Mexico City, Mexico

ABSTRACT

PASION is a process- and event-oriented simulation language designed for PASCAL users. The language has a two level (process/event) structure and permits the use of all the Pascal structures. It also offers the main features of object-oriented programming. PASION provides necessary facilities to handle sequences of random events, queues and quasi-parallel processes, both discrete and continuous.

1 WHY A NEW LANGUAGE ?

A look at the annals of simulation software development could result in the impression that we have enough (or, rather, too many) simulation languages and packages. The idea of creating a new one might appear to be crazy. However, looking at the existing simulation software and at the recent tenden-cies in programming, it can be seen that the existing simulation software becomes somewhat obsolete. The new simulation software should be object-oriented not only because almost all new software is object-oriented (see Schmucker, 1986, for a review on object-oriented programming). The simulation software must have this orientation simply because *the real world is object-oriented.* Second, it is not a proper way to develop good simulation tools modifying or extending languages which are 20 or even 30 years old. In my didactic work I have been looking for a well structured and easy-to-teach simulation language. It seems that the most complete one is Simula. However, it is somewhat difficult to teach Simula quickly, for its relation to Algol. Similar difficulties appear when using Modula-2. Shortly speaking, the new language should be object-oriented, and should offer the following features: clear process/event structure, efficient clock mechanism, com-bined continuous/discrete modeling and inheritance. It also should have an environ-ment which supports graphics, interactive simulation and auxiliary modules (program generators) for queuing and continuous models.

2 THE LANGUAGE

Let us recall some basic concepts of process-oriented simulation. To describe a sequence of events we must specify event operations and describe both dependence of each event on the model time, and the interactions between the events. A process-oriented language offers something more. Namely, it defines a structure within the set of events by introducing different processes. By the process we mean a generic program segment which declares a specific object type. This declaration describes the properties of objects which can execute events in relation to the model time. According to this decla-ration the corresponding objects can be crea-ted at run-time. This approach makes the simulation object-oriented . Recall that an object-oriented language should support information hiding, data abstraction, dynamic binding and inheritance (see Dahl and Nygaard 1967, Koehler and Patterson 1986a, 1986b; Pascoe 1986; Schmucker 1986). PASION does not fully implement all these concepts. It sup-ports dynamic creation of objects, data hiding and, to some extend, data abstraction by the mechanism of predefined processes and inheritance. The following example shows a PASION program where two objects activate each other.

```
PROGRAM TRIGGER;
REF A,B:X;
{A and B point to objects of type X}

PROCESS X,2; {A process type}
ATR N:STRING[6];
```

```
EVENT ONE; {ONE is an event}
WRITELN('Active object: ',N);
IF THIS=A THEN B.ONE:=TIME+1.0
ELSE A.ONE:=TIME+1.0 ENDEV;

START {Main program}
NEWPR A; A.N:='FRED';
NEWPR B; B.N:='ANDREW';
A.ONE:=TIME;
{The object A starts immediately.
B waits.}
$ {This terminates the program.}
```

This is a complete PASION program. The reference variables A and B are used to refer to the two objects of type X. The instruction A.ONE:=TIME+1.0 schedules the event ONE of the object A to be executed at TIME+1.0, where TIME is the model time. THIS is a reserved word which can be used to refer to the object from within its scope. Thus, THIS=A is true when the object is referred by A and false for the object referred by B. It can be seen that the two objects activate each other. The output from this simple program is as follows.

```
Active object: FRED
Active object: ANDREW
Active object: FRED
Active object: ANDREW
Active object: FRED ..... etc.
```

It is a very simple example. In practice, the simulation programs can have up to 50 process declarations (object types) and up to 400 events in each process. The number of objects generated at run time depends of the complexity of the objects. Models with more than 2000 objects were successfully run on the IBM XT.

3 CORRESPONDENCE BETWEEN MODELS AND PASION PROGRAMS

According to the commonly used simulation terminology (see Zeigler, 1976) a simulation model is composed by its components (e.g. clients in a shop). The state of each component is described by the corresponding set of descriptive variables and its activi-ties are given by the rules of interaction between the components. Experimental frames define the actually used set of descriptive variables and determine the complexity of the model. The PASION language has all these basic modeling elements. Model components are ob-jects, component specification is given by the process declaration, descriptive variables are process attributes and the component ac-tivities are events. Experimental frames can be expanded using the mechanism of inheri-tance. Inheritance enables programmers to create classes and therefore objects that are specia-lizations of other objects. This enables the programmer to create complex models by using code created and tested before. Inheritance in PASION can be applied using prefixed process declarations. Let PA be the name of an existing process. Suppose that we wish to create a new one, say PB, having all the properties of the process PA (this means all its attributes and events). This can be done using the name PA/PB instead of PB in the heading of the process declaration. While processing such declaration, the translator looks for the process PA (the parent process) and inserts all the attribute declarations and event descriptions from PA into the new process PB (derived process). Parent processes can reside in separate files, or be placed in the same source file. Thus, the user can prepare and store some useful source "capsules" and use them while creating new processes. During the creation of the new (derived) process some of the names used in the parent process can be changed. This includes variable or type identifiers.

4 SOME EXAMPLES

Let us consider a simple example of object-oriented simulation in PASION. To simulate the growth of a plant it is suffi-cient to describe the behavior of one "cell" of the plant. It can be a "branch element" which can generate one or more other branch elements which grow upwards, with random inclination. The branch element can also increase its thickness to "support" more bran-ches. The program describes one branch element with two events: "generating new branch" and "to get fatter". The main program generates one initial core element which generates the branches (other objects of the same type), which, in turn, generate other branches etc. It is easy to show this process on the screen, as indicated in Fig.1.

Observe that this simulation is not only the generation of the image of the plant. Each "branch element" of the plant is "alive", being an active object of the model and its behavior can be modified in order to experiment with the model. As an example of another object-oriented simulation consider a two dimensional heat

Fig1. Simulation of a growing plant.

distribution problem. Suppose that the heat propagates in a rectangular plane section. Let us discretize this region replacing it with a uniform red of points, without defining any time-discretization. Each point is an object in the simulation program. The attributes of an object are its coordinates, its temperature and the proper heat. The only activity of each object is to adjust its tem-perature according to the temperatures of the four nearest points and according to the heat conductivity of the material between the points (not necessarily the same for different points). The fixed boundary conditions in this model can be defined by fixing the temperature for some objects (disactivate them). The "free boundary condition" for the points at the boundary of the region consist in the fact that these points have only three and not four neighbors. The activities of the objects are programmed to be executed in random time intervals, so that no fixed time-step exists in the model. The simulation consists in gene-rating the points and activate them. The steady state reached by the program gives the solution to the simulated heat distribution problem. The stability of such algorithm depends on some additional model parameters, not discussed here. Fig. 2 shows a solution where the region consists of 400 points (20x20 net). The point (13,13) is a heat source with constant temperature and the point (3,3) is an ideal heat sink. The boundaries of the region are isolated from the environment (free boun-dary conditions). The figure shows the final, steady-state situation. The dynamics of the simulated heat propagation is given by the evolution of this plot in the model time.

Simulation of such kind of the distributed-parameter systems is rather slow and not so efficient as the solution of the corresponding partial differential equation. On the other hand, observe that the object-oriented model does not involve any differential equation at all. Consequently, we are not restricted by regularity assumptions which are rather strong when the partial equations are considered. With little modifications the model can describe any strongly "irregular" system, when, for example, the properties of the material are discontinuous functions of the temperature or when two or more similar non-continuous processes interact with each other as occurs in alloy solidification problems, quite difficult to simulate using differential equations.

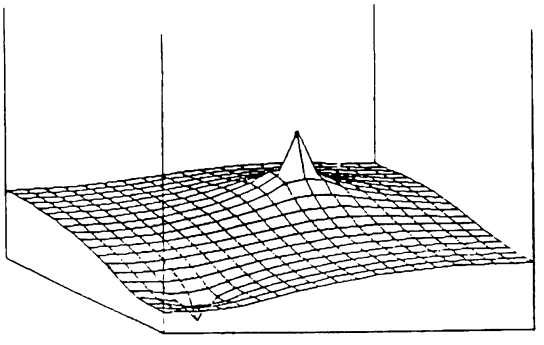

Fig.2. Heat transfer simulation.

5 PASION ENVIRONMENT

There are few programming languages which can be effectively used without an appropriate environment. PASION is equipped with the Minimal PASION Programming Environment (MPPE) which consists of a library of predefined processes and other modules. It supports interactive simulation, graphics, statistical analyses of the results, continuous dynamic models and queuing models. The core of MPPE consists of the library of PASION predefined processes. These are generic program segments which generate process declarations (not objects). Predefined processes are written in PASION extended by a simple "meta-language" which permits a process to have formal parameters. The user invokes a predefined process by its name and specifies the actual parameters, which are passed to the corresponding process declaration in the user program by name, be-fore the program is translated to PASCAL. These parameters can represent not only variable names but also types, complete ex-pressions, instructions, comments etc. The user can prepare his own predefined appli-cation-oriented processes and add them to the library.

6 QUEUING MODEL GENERATOR

PASION has a predefined type QUEUE which is a line of type FIFO, LIFO or RANDOM. It offers a number of procedures and functions to handle lines in the queuing models. Thus, the user can declare some queues and code operations on them. Other mode to simulate queuing models is provided by a module of the MPPE named Queuing Model Generator (QMG) which makes it possible to simulate systems with queues without any programming, using a graphical model description provided by the user. When applied to queuing models, QMG offers nothing more then GPSS or SLAMII which are, perhaps, better packages to such applica-tions. Observe, however, that in some situations it has certain advantage. For example, the flexible manufacturing systems (FMS, see The Charles Stark Draper Lab, Inc., 1984) are controlled by quite complex algo-rithms and run in a complex computerized environment . Consequently, any package used to simulate FMS must be embedded in an appropriate programming environment. QMG satisfies this requirement, being related to an algorithmic, object-oriented language (Ra-czynski 1990).

7 CONTINUOUS MODEL GENERATOR

This program was designed in order to facilitate simulation of dynamic continuous systems. The Continuous Model Generator (CMG) is a program generator which generates source PASION and/or PASCAL code, according to the model specifications given by the user, mainly in graphical form. The CMG output is created in the form of a PASION process declaration which can be inserted into any PASION (continuous, discrete or combined) model. CMG also can generate a complete PASCAL program which can be run using a PASCAL compiler. The input to CMG is formu-lated in terms of graph diagram which describes the dynamics of the modeled system. By the graph diagram we mean a network com-posed of nodes and directed links. Nodes represent signals and links represent transfer functions. CMG permits the following types of links: Static linear, Static non-linear, Dynamic linear (given transfer function), Time delay, Sample-and-hold and Superlink (a complex dynamic system). The last link type (Su-perlink) permits to include whole dynamic model (specified earlier and stored in a file) to the model actually being created. This feature is useful while developing complex models, composed by submodels created and tested separately. When simulating combined (discrete/-continuous) systems it is possible to declare "state events" i.e. events which occur when some continuous state variables reach a specific level.

8 IMPLEMENTATION

PASION-to-PASCAL translator runs on the IBM PC and compatibles. The "6000" version is being developed for the IBM RISC RS-6000 computer. The code produced by the translator must be compiled by a PASCAL compiler. The resulting program expands dynamically while new objects appear, so that the number of objects which can run simultaneously depends on the amount of the operational memory available at the run time and on the size of data blocks (attributes) of the objects. PASION has been used in teaching simulation methods. It is important to have an easy to learn simulation tool which may be used to illustrate the concepts of process declarations, objects, events, inheritance, preprocessing and animation, when the students have some knowledge on structural programming in PASCAL and does not have any experience in simulation.

REFERENCES

The Charles Stark Draper Lab, Inc., 1984 "Flexible Manufacturing Systems Handbook", Noyes Publications.

Dahl O., Nygaard, 1986, "Simula - An Algol-based Simulation Language", Communications of the ACM no.9, pp.671-678.

Koehler T., Patterson D., 1986, "A Small Taste of Smalltalk", BYTE 11(8), August 1986, pp.145-159.

Koehler T., Patterson D., 1986 "A Taste of Smalltalk", W.W.Norton & Co., New York.

Pascoe G.A., 1986, "Elements of Object-orien ted Programming", BYTE 11(8), August 1986, pp.139-144.

Raczynski S., 1986, "PASION - Pascal-related simulation language for small systems", SIMULATION 46(6), June 1986, pp.239-242.

Raczynski S., 1987, "PASION, Lenguaje para Simulacion: una Introduccion", Computer-World/Mexico, no.178,179,180, April-May 1987.

Raczynski S., 1988, "On a simulation experi ment with a parallel algorithm for optimal control", Transaction of the Society for Computer Simulation, vol.5, no.1, pp.87-97.

Raczynski S., 1988, "Process hierarchy and inheritance in PASION", Simulation 50(6).

Raczynski, S., 1990, "Graphical description and a program generator for queuing models", Simulation 55(3).

Raczynski S., 1991,"Simulacio'n por com putadora - metodos y lenguajes",LIMUSA.

Shmucker K.J., 1986, "Object-oriented Langua ges for the Macintosh", BYTE 11(8), August 1986, pp.177-185.

Zeigler B.P., 1976, "Theory of Modeling and Simulation", John Wiley & Sons.

AUTHOR BIOGRAPHY

STANISLAW RACZYNSKI received his Ph.D in automatic control from the Academy of Mining and Metallurgy in Poland, 1969. He has been working for several years at the same Academy, at the Institute for Control Problems in Moscow USSR and at the National University of Mexico. Recently he is a professor at the Pan- American University in Mexico City. He is an active member of SCS, working in the area of control and simulation methods.

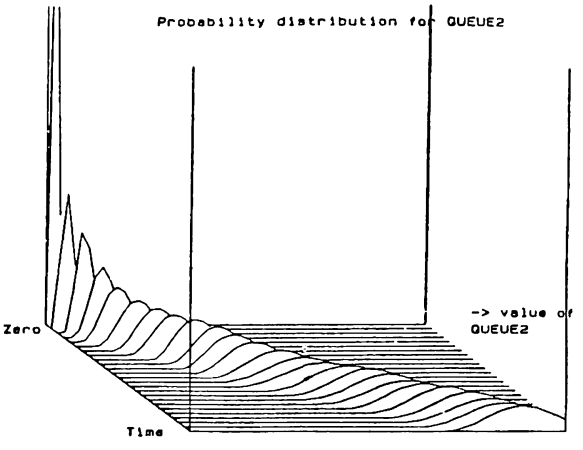

Fig.3. Exapmle of a QMG output. The 3D image of the probability density function for a queue.

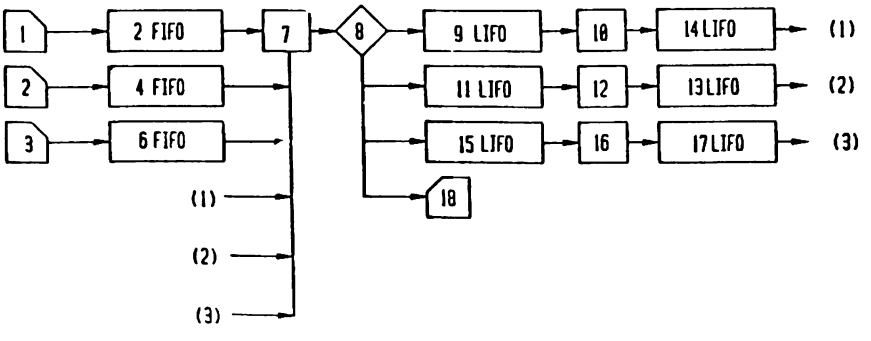

Fig.4. An example of a QMG scheme. 1,2 and 3 are input streams, 2,4,6,9,11,15,14,13, and 17 are queues, 7,18,12 and 16 are servers.