Symulacja w Badaniach i Rozwoju
Vol. 3, No. 4/2012

Stanislaw RACZYNSKI
Uniersidad Panamericana
Augusto Rodin 498, 03910 Mexico City, Mexico
E-mail: stanracz@stanr.com

Object Properties in Discrete Event Simulation: Attributes, Functions and non-typical Behavior

1 Introduction

Important questions both in teaching and implementing computer simulation are:

- Must a simulationist be a programmer?
- Must he/she be a mathematician?

Depending on what is the role of the simulationist in the whole process of creating and using a simulation program the response may be yes or not. It seems that the commercial simulation tools are being developed in order to prevent the user from any coding and to make all the mathematics (statistical considerations) as transparent and simple as possible. To say that this is a correct tendency, first of all we must know who the simulationist is. If he/she is an industrial engineer, a sociologist, a warehouse manager or just an amateur, the tool should be "fast and easy", and the simulationist need not be a programmer or a mathematician (**user of kind 1**). However, if he/she is a professional simulationist who develops simulations in serious and professionally advanced way, he/she should be able to create and code necessary algorithms and be aware of the model mathematics (**user of kind 2**). While teaching computer simulation it is not always clear to which kind of users we should address. This is an important question, because the contents of the simulation course and the tools used, strongly depend on the student/user kind.

It seems that the new discrete event and general purpose packages are being designed mostly for the users of kind 1. The new software has always a well designed Graphical User Interface (GUI) and offers ready-to-use encapsulated probability distributions and statistics. However, despite of good manuals and additional materials, the users of kind 1 frequently commit fundamental errors, like using the Poisson enter-arrival time distribution for the Poisson arrival process.

One could say that users of kind 2 can do their simulations in any algorithmic language and need no simulation packages. Obviously, this is not true. A simulation tool should provide ready to use mechanisms to avoid unnecessary work (but nothing more). In general, the question is if, in our simulations, we need intelligent objects (agents) at all. The answer is yes. Such objects are not needed in academic examples and simple simulations (users of kind 1). However, if we face the reality, for example a real manufacturing system, it is quite sure that there will be objects that do not fit in standard blocks or facilities offered by most of the simulation packages. In such case the use of intelligent objects is inevitable.

This paper is not a survey of simulation software. The packages mentioned below are only selected examples. It is supposed that the reader already knows how they work.

2 Time and event management, and object behavior - some known tools

By *Time and Event Management* (TEM) we will understand the model time clock and event queue management, including the basic queuing model operations, provided by the simulation package. The *Object Behavior Modeling* (OBM) is a set of additional items like user-defined distributions and logical functions, non-typical operations, object attributes and the general object behavior.

Let us start with GPSS (*General Purpose Simulation System*), omitting earlier tools like a forgotten but very nice language of 1950s CLS (*Control and Simulation Language*).

2.1 GPSS

This language, developed primarily by Geoffrey Gordon at IBM around 1960 [2], has contributed with important concepts. This is an old tool, but it is still used and works perfectly. In fact, GPSS is an object-oriented tool, although it does not fit into the modern object-oriented paradigms. The objects in GPSS are called *transactions*. These are moving items that appear, go through the fixed model facilities and disappear. GPSS World has been extended by PLUS, the *Programming Language Under Simulation*. The TEM level instruction set of GPSS is simple, easy to use and works perfectly. It can be dominated by anyone in few hours of learning. The OBM level mechanisms are not so easy. If the user wants to equip objects (transactions) with any additional properties and individual, non-standard behavior, he must learn PLUS and dominate the information about the SNAs (*Standard Numeric Attributes*). The Plus manual is a whole chapter of the GPSS manual, or a separate document of about 60 pages. The SNA documentation occupies also several dozen pages, including such items as A1, AC1 (absolute clock), C1, CAx, CCx, CHx, CMx, CTx, Fx, FCx, FIx, FNx, FRx, FTx, FVx, GNx, GTx, LSx, MBx, MPx, MXx, M1x, Nx, Px and about 30 more (x is usually an integer number). Learning all this stuff the user can simulate more advances models, but the created objects can hardly be considered as "intelligent".

2.2 ARENA

Arena modeling system from Systems Modeling Corporation is a nice and widely used simulation tool. It is equipped with a GUI and animation mechanisms [3]. The TEM level of Arena permits to quickly create a queuing or manufacturing discrete event models. No coding is needed and the task results in clear flowcharts of the model. The OBM level is somewhat more complicated. Arena is built over the SIMAN [5] simulation language. So, first of all, to be able to manage somewhat more sophisticated object behavior, the user must learn SIMAN. This permits the use of user-defined logics, statistics and/or a non-standard object behavior. The Arena entities (moving objects) can be equipped with *attributes* of the following type: *General attributes, time attributes, cost attributes, entity type variable, group member variables* and *other*. The specification of the attributes and other Arena pre-defined variables takes about 30 pages in the Arena documentation. Again, if the user wants to create and manage a little bit more complicated object behavior, he/she must learn SIMAN and dominate dozens of pages of the Arena manual.

2.3 SIMIO®

This is a multi paradigm software delivered by Simio LLC. Simio is created by a team of simulation software developers led by Dennis Pegden [5] [6].

Compared to Arena SIMIO is a step forward in creating models with intelligent objects. The object definition in SIMIO is more general. Objects may be fixed facilities or moving dynamic objects named entities. The user can define his own objects, store and reuse them, or use the objects from a standard library. These may be: fixed (server, machine), link (a pathway for entities), node (link intersections), entity (dynamic object, like client in a shop), transporter (it can pick up and drop entities at nodes).

The user defines the object properties. They may be of different types such as strings, numbers, selections from a list and expressions. The properties are edited in multiple edition windows. There are many ways to define a SIMIO model. A programmer familiar with an object oriented language like C++ or Delphi can understand and dominate the SIMIO modeling in reasonable time and effort. SIMIO creators claim that the process-based objects in the SIMIO model are both simpler and more powerful than the code-based objects in other modeling tools. SIMIO offers both TEM and OBM facilities, although they are not clearly separated from each other.

2.4 SIMULA

We must mention here Simula [1] (its mostly known version 67). Although it was a tool developed more that 50 years ago, it is still perhaps one of the most advanced and elegant object oriented languages. In fact, Simula itself is just object-oriented and not a simulation language. The modeling facilities have been added to the language as a part of its standard class library, and are encapsulated in a pre-defined class named Process. Any object that inherits the Process class properties can use the clock mechanism, and event scheduling and execution mechanisms (called here TEM). The OBM (object behavior management) is coded directly in the language. As an old software, it originally had no GUI and other graphical facilities. The language is rather difficult to learn, and needs previous training in Algol.

One of the recent trends is to enable the user to create intelligent objects in discrete event simulation. First of all, we should agree what means "intelligent". If we want the objects to be able to add and multiply numbers and execute logical expressions, most of the simulation packages provide these features. However, if we define the "intelligence" as the ability to make decisions due to a more sophisticated algorithms or equip the objects with some kind of artificial intelligence, only advanced object-oriented algorithmic languages provide such features. Looking at the above example packages, only Simula has this capacity.

Anyway, if someone wants to create an object-oriented simulation package with intelligent objects, he/she finally must create a new high level object-oriented algorithmic language, even if it runs behind a well designed GUI. The question is: isn't it better to take a known, complete, widely known, used and advanced language and add to it the time and queuing management layer (TEM)?

3 BLUESSS/QMG concepts

Let us compare the software mentioned above with some concepts of the BLUESSS (Blues Simulation System). The comparison refers only to the BLUESSS queuing module (QGM) and not to the whole system.

BLUESSS evolved from the PASION and PSM++ packages, related to Delphi. Some applications and remarks on discrete event simulation can be found in [7], [8] and [9]. The package runs over the Borland's C++Builder. The user can be of kind 1 or of kind 2 (programmer skills). Taking about a professional simulationist we should rather think about users of kind 2. My point is that few really professional simulationists do not dominate C++. BLUESS is a simple simulation language and has the BLUESSS-to-C++ translator. The user can code his model in the BLUESS language, or use one of the BLUESS modules to create models without coding.

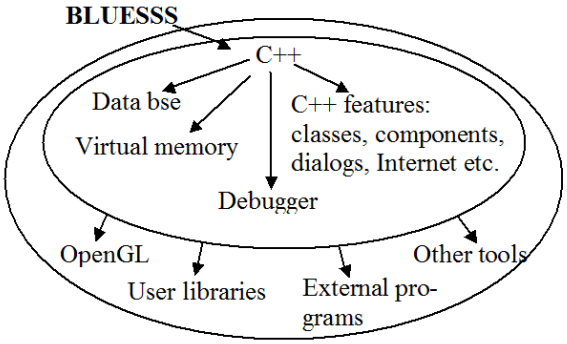

Fig. 1. BLUESSS features

The following modules (source code generators) are included in the BLUESSS package:

*** Queuing Model Generator**

*** Flow diagrams.** Continuous simulation using signal flow diagrams.

* Continuous simulation using **bond graphs**.

* Continuous simulation, **ordinary differential equations.**

After defining the model, the BLUESS system generates the C++ code and invokes the C++Builder which produces the executable program. The event queue in BLUESSS works due to the three-phase discrete simulation strategy [4].

Using the QMG module the user defines the model in the Arena-like style, with no coding at all. As the process of creating exe file (stand-alone, independent executable) passes through the C++ compilation, the BLUESSS QMG module can use all the features of C++ (see figure 1).

In other words, the comparison of QMG with, for example, Arena can be summarized in the following table:

Tab. 1. Creating intelligent objects (Arena vs. BLUESSS)

Arena	Why not replace it with:
Create model flowchart with Arena	Use QMG graphical model editor
Learn SIMAN, learn Arena manual including 30 pages of entity attribute and expression specifications, code the necessary expressions	Use C++

Note that the QMG graphical model editor is very simple and can be dominated in 15 minutes of "training", even without consulting any documentation. The objects created in QMG can be equipped with a simple abilities (logical expressions, additional attributes). However, in the advanced simulations they can be equipped with more complicated behavior, like decision making algorithms, fuzzy logic, iterative optimization algorithms, neural nets, database consulting and other. They can execute external programs or use external files. The object can do everything what can be coded in C++. There are no restrictions on the type and size of its attributes (those can be numbers, strings, arrays, pointers, and/or C++ structures of any kind). If required, the object can communicate through the Internet, sing a song, display an OpenGL image, execute an external program etc. Obviously objects cannot intervene in the TEM (Time and Event Management) of QMG. There are also some restrictions on the use of pointers.

To create a QMG flowchart the user picks up blocks (like GPSS facilities or Arena modules), and defines the model structure and the basic block parameters like inter-arrival times, service times etc. Then the simulation can be invoked. The entities (dynamic objects) appear, go through the blocks and disappear. As stated before, the additional entity attributes can be declared, being of any available C++ type.

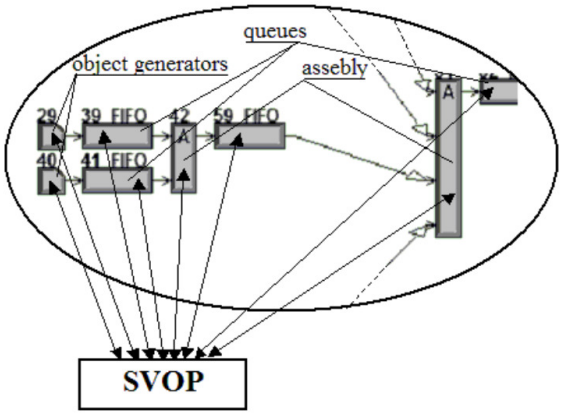

Fig. 2. Fragment of a QMG model. Automatic calls to the SVOP function

The relation between the entities and C++ entity-related code is very simple. Any entity which enters to any of the model blocks, calls a global C++ function named *SVOP*. Both calling entity and block identifiers, as well as all entity attributes are passed to *SVOP* as actual parameters. For the assembly operation *SVOP* is called by each entering entity and for the new (assembled) one. So, in the *SVOP* body the user can identify the block/entity pair and code any required action. For example, entities can enable or disable model blocks (using the *semaphore* logic variables) or execute more complicated actions (figure 2.).

Suppose, for example, that we need the following actions to be taken:

* If an entity enters the queue 59, its string attribute *myname* includes the string "*dog*" and the entity age is greater than 100 model time units (the time spent in the system), then invoke external program *other.exe*.

* If any entity enters the assembly block 42 and the sum S of the length of queue 39 and queue 41 exceeds 24, close (disable) generators 29 and 40. If S is less or equal to 24 then enable these generators.

* If any entity waits in any queue for more than 20 time units, display a warning message

In the below code *n* is the calling block number, *SOURCE* is the number of the block where the entity has been created, *TIMIN* is the model time instant when the entity has been created, *TIMQ* is the time the entity has been waiting in a queue (if it is actually waiting), and *myname* is an additional, use-defined entity attribute. *TIME* is a global variable representing the model time. The function *nr* returns the queue length. *QUEx* is the reference to the queue block number *x*, and *SEMx* is a boolean variable (a semaphore) that enables (if true) or disables (false) the block number *x*. *DisplayWorning* is a user-defined C++ function (may show something on the screen, emit a sound etc.).

The *SVOP* procedure in this case may be as follows:

```
void SVOP(int n, int SOURCE,
  float TIMIN, float TIMQ, String& myname)
{
bool b;
if(n==59 && AnsiPos("dog",myname)>0
   && TIME-TIMIN>100)
   WinExec("other.exe",1);
if(n==42){
  b=nr(QUE39)+nr(QUE41)>24;
  SEM29=b; SEM40=b;}
if(TIMQ>20)DisplayWarning();
}
```

This is a very simple example. Inside the *SVOP* function the user can insert any C++ code to define the entity behavior and/or block operations.

4 More about BLUESSS

As stated before, BLUESSS is general purpose package. It contains several modules (source code generators) for queuing/manufacturing models, continuous simulation using ordinary differential equations, signal flow graphs, bond graphs or combined models. The user can create the source code or use any of the BLUESSS modules to avoid coding. Any BLUESSS simulation task passes through the source BLUESSS code and C++ code generated automatically. The final product is an independent *exe* file, ready to run. The package structure is shown on figure 3. In BLUESSS everything (except the code taken from C++ libraries) passes through the BLUESSS source code and through the C++ code. The user can create his/her code or use the code generators. Other options are:

Queuing models: The queuing module generates the source code which is translated to C++ and compiled.

ODE (Ordinary Differential Equations) module receives the right-hand-sides of the equations. The rest is done automatically (source code generation, compilation).

Block diagrams and signal flow module: The user defines graphically the model structure and the necessary parameters. The module generates the model equations, the rest is done as above.

Bond graphs: The user draws the bond graph model and gives its parameters. The rest is done automatically.

Animator: 2D off-line animation of queuing models is available (see figure 4).

Variance analysis: post-mortem additional statistical analysis can be invoked. This includes the max-min and confidence intervals for the model trajectories, shown as functions of time. This feature, provided by few simulation packages is very useful while simulating queuing and stochastic models. On figure 5 an example of such plot is shown. This is the length of a simulated queue. The gray region is where the length of the queue is supposed to belong with probability 0.92. The curve inside the region is the average queue length in function of model time. The average is taken over a series of

repeating simulations. If the gray region is big (big variance), then it can be seen how little information provides the average value.

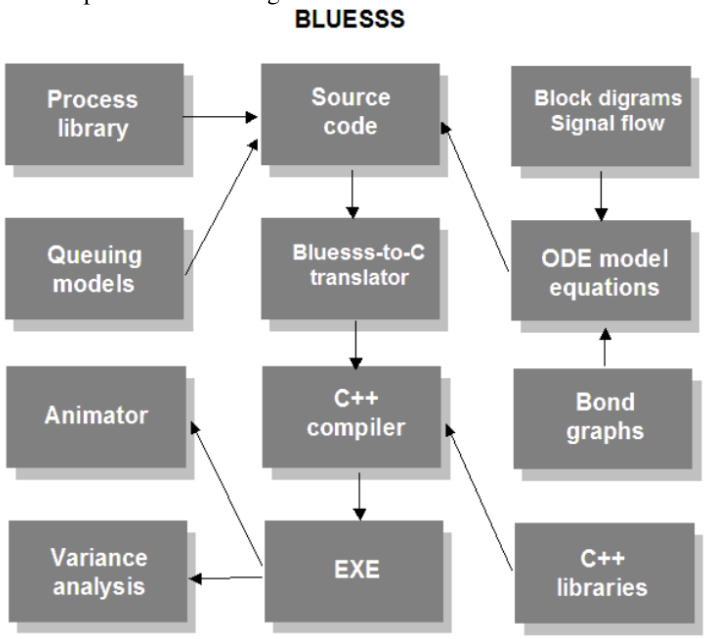

Fig. 3. BLUESS package

The user can see and modify both the BLUESSS and C++ codes. For queuing models, he/she can also use the SVOP function as described earlier. Although the queuing models of BLUESSS are rather simple, the possibility of working on the generated code makes it possible to simulate any required object behavior.

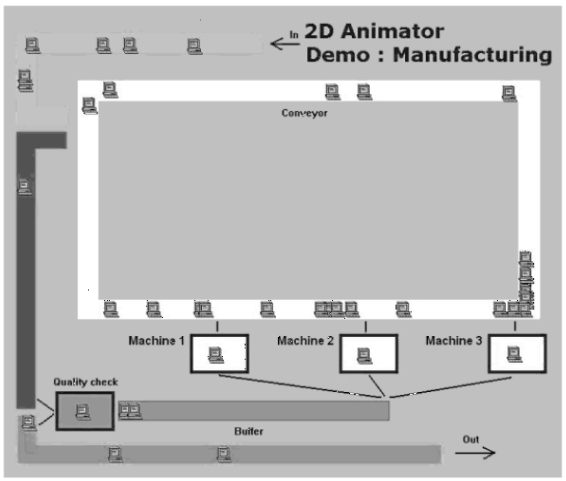

Fig. 4. Bluesss animation of a manufacturing model. Queues, machines and conveyers

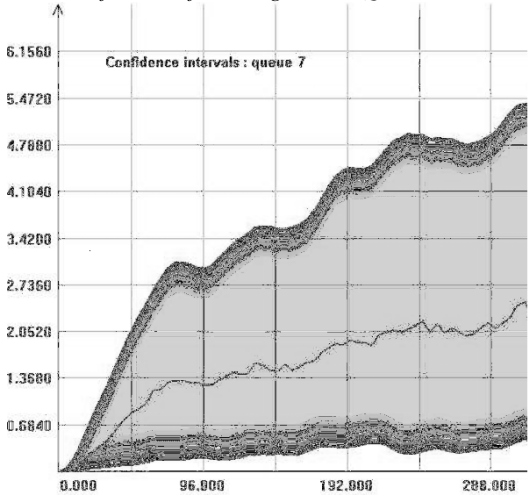

Fig. 5. Confidence intervals for the length of a queue as a function of time

Those are only some examples of BLUESSS features. BLUESSS can use any tools available in C++. Interesting animations, both for continuous and discrete event models can be created using the OpenGL graphics. Some tools that facilitate the BLUESSS-OpenGL communication are provided in the package.

Observe (figure 3) that both discrete event and continuous models result in the BLUESSS source code. The only difference is that the continuous models are simulated as a sequence of events with a small time step, each event being a call to one of the possible numerical methods for ODE. This means, that at the source code level the user can mix discrete and continuous models in the same simulation program. On the figure

6 you can see a screen of a continuous model simulation (multiple pendulum), animated with OpenGL graphics.

5 Conclusions

BLUESSS is not a widely know commercial software. This is rather a proposal on a possible direction in simulation software development. The point is that what is really needed in discrete event simulation consists in the fast clock machine (event queue management) and good model structure definition tools.

The algorithmic aspect of the modeling, like intelligent object behavior, should be handled by a high level algorithmic language rather than complicated parameter specification and expression building incorporated in the package. Using one of already developed and powerful object-oriented language in the background gives us a much more versatile tool to handle all what is not a simple queuing and discrete event simulation. In the case of professional level of simulation tasks, there are few developers who don't already dominate C++ or similar tools and should be able to use these abilities in modeling and simulation.

References

1. Dahl O., Nygaard B.: Simula - an Algol-based simulation language, *Communications of the ACM*, no.9, pp.671-678, 1967
2. Gordon G.: *The application of GPSS to discrete system simulation,* Prentice-Hall, 1975
3. Kelton D., Sadowski R and Sadowski D.: *Simulation with ARENA*, McGraw- Hill, New York, 2004
4. O'Keefe R.M.: *The three-phase approach: A comment on strategy-related characteristics of discrete event languages and models*, SIMULATION 47(5), pp. 208-210, November 1986
5. Pedgen, C. D., Shannon R. E. and Sadowski R. P.: *Introduction to simulation using SIMAN*. McGraw Hill, 1995
6. Pedgen C. D. and Sturrock D. T.: *Introduction to Simio*, in *Proceedings of the 2010 Winter Simulation Conference*, Simio LLC, Sewickley, PA, USA, 2010
7. Raczynski S.: *Alternative mathematical tools for modeling and simulation: Metric space of models, Uncertainty, Differential Inclusions and Semi-discrete Events*, Proceeedings, European Simulation Symposium ESS2000, Hamburg, Germany, September 2000
8. Raczynski S.: *Simulation of the Dynamic Interactions Between Terror and Anti-Terror Organizational Structures*, The Journal of Artificial Societies and Social Simulation, vol.7, no.2, England, 2004, http://jasss.soc.surrey.ac.uk/7/2/8.html
9. Raczynski S.: *Modeling and Simulation: Computer Science of Illusion*, Wiley, England, 2006

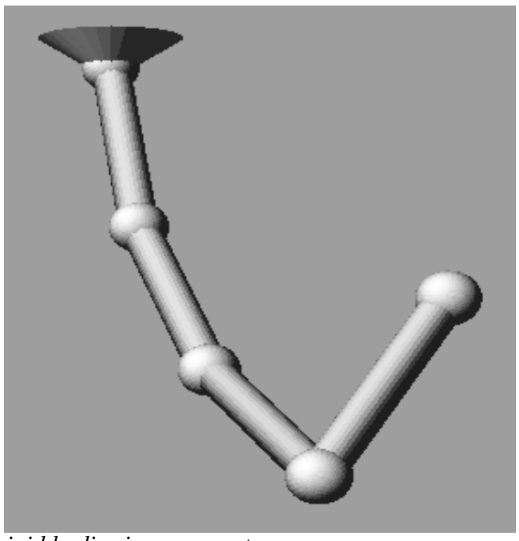

Fig. 6. A chain of rigid bodies in movement

Summary

A comparison between old and new standards like GPSS, Simula67, Arena, Simio and BLUESSS is made from the point of view of additional user-defined object attributes and related functions. The scope of the applications under consideration is limited to the discrete-event, queuing and manufacturing simulation. The focus is made on the management of the object attributes and related operations, and on the way to make the simulation tool flexible enough without complicating the basic model building operations. A proposal of a new simulation tool is presented.

Keywords: PSM++, BLUESSS, Simula67, GPSS, Arena, Simio,C++, simulation, queueing, manufacturing