Revitalizing Object Oriented Discrete-Event Simulation with The Simula and Demos Philosophy

José M. GARRIDO
Department of Computer Science and Information Systems, Kennesaw State University
Kennesaw, GA 30144, USA

ABSTRACT

After more than forty years since the key concepts of object orientation were first introduced in Simula, these concepts have evolved since then and have been implemented in most modern programming languages. Objects allowed the process view of modeling the world and this may be considered the best way of developing discrete event simulation models. The DEMOS package added more powerful simulation facilities to the Simula language. However, Simula and DEMOS never became widely used, despite the significant contribution to object orientation and simulation.

The Object Oriented Simulation Language (OOSimL) is introduced as a modern language for developing discrete-event simulation models using the object-oriented and process modeling approaches. Part of the syntax and semantics of OOSimL was influenced by Simula and the DEMOS package.

OOSimL can also be used effectively as a higher-level object-oriented general-purpose programming language. It supports the general conceptual frame-work of the process style of discrete-event simulation and that facilitates the teaching and learning of object-oriented modeling and simulation.

Keywords: Object orientation, Modeling, Discrete event simulation. Simula, DEMOS, Programming.

1. INTRODUCTION

The SIMULA [1] language was developed by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Center (NCC) in Oslo around 1962. It was originally designed as a language for discrete event simulation, as an extension of the programming language ALGOL 60; it was later extended and reimplemented as a full scale general purpose programming language. Simula was the first language that introduced the concepts of classes and objects. The language also introduced the idea of prefixing and subclasses.

In the early 1970's, Alan Kay and his group at Xerox PARC used Simula concepts as a platform for their development of Smalltalk, extending object-oriented programming importantly by the integration of graphical user interfaces and interactive program execution.

In the mid 1980's, Bjarne Stroustrup started his development of C++ by bringing the key concepts of Simula into the C programming language. Other object-oriented programming languages such as Eiffel, CLOS, Java, and others, were also influenced by Simula.

Simula 67 introduced the notion of modeling a dynamic system as a set of interacting quasi-parallel processes. These processes were implemented as coroutines [2], [3].

Demos was developed by Graham Birtwistle [4] and extended Simula with a library of standard process synchronization devices together with facilities for data collection, random number generation, tracing and automatic reporting.

The simulation aspects of Simula and DEMOS may seem to have not directly been implemented in

modern simulation languages. However, some of these facilities have been implemented in a few simulation packages that are used with Java or C++.

2. THE OOPSIM PROJECT

The OOPsim project of the Department of Computer Science and Information Systems, Kennesaw State University, has as its main purpose, to investigate improved methods and tools for the education of object oriented modeling and process approaches to discrete-event simulation.

One of the outcomes of the OOPsim project is the design and development of the general-purpose object oriented and discrete-event simulation language, OOSimL. Our view is to consider OOSimL as the Object-Oriented Simulation Language for modeling and implementation of discrete-event simulation models in education.

The project currently provides the compiler for the new OOSimL general-purpose object oriented simulation language, the PsimJ2, PsimJ [5], [6], and Psim3 simulation packages, and document files.

3. THE OOSIML LANGUAGE

The general-purpose simulation language OOSimL supports the object-oriented approach to modeling and the process interaction approach to discrete-event simulation. This allows for higher-level simulation modeling and the flexibility of allowing as much details as necessary, depending of the level of abstraction in the model.

The purpose of this language and software tool is to support a conceptual framework based on the early introduction to abstraction, object-oriented modeling, logic, discrete-event simulation, and other concepts in problem solving and software engineering.

The software development approach used with OOSimL includes modeling with UML, DEMOS Activity Diagrams with the Pooley/Hughes extensions, implementation with flowcharts and pseudo-code. The overall goal is to provide an easier way to understand problem solving and the development of working programs.

To support the general conceptual framework mentioned previously, OOSimL, was developed as a high-level object-oriented programming and simulation language that facilitates the teaching and learning of the programming principles emphasizing object-oriented modeling.

The notation in the language is very high-level and includes conventional pseudo-code syntax that is much easier to read, understand than the syntax of standard object-oriented programming languages such as C++ and Java.

The OOSimL compiler carries out syntax checking and automatically generates an equivalent Java program. The language has the same semantics as Java, so the transition from pseudo-code to Java is, hopefully, very smooth and straightforward.

Thus, the first overall goal is to help students reason about the problem at hand, design the possible solutions to the problem, and write programs that are: easy to write, easy to read, easy to understand, and easy to modify.

The second goal is to consider OOSimL as a tool for better understanding and learning about the object oriented design, implementation, and running of simulation models of small, medium, to large and complex systems.

Programs written with OOSimL can be easily integrated with Java programs. Therefore, any Java library can directly be accessed by a model written in OOSimL. The language supports the reuse of Java components. The run-time support of this simulation language is based on the PsimJ2 simulation package, which is a collection of Java classes and is used to run simulation models.

The simulation modeling in OOSimL follows the philosophy of Simula and DEMOS. A dynamic system is modeled as a set of interacting processes. All the process states and the language statements for the timing, sequencing, and synchronization facilities for manipulating processes are present in OOSimL. A process is implemented as a thread object, instead of a coroutine, as in Simula. The other components of a system are modeled as conventional (non-thread) objects.

4. BRIEF COMPARISON WITH SOME OTHER LANGUAGES

Our experience has been until very recently, using the PsimJ2 simulation package [2] with Java, and the Psim3 simulation package with C++. Students are normally required to have a fairly detailed knowledge of either Java or C++ programming languages, or both. On the other hand, most commercial simulation software systems (ProModel, Arena, Network, etc) are actually application-specific tools for developing simulation models. Most of these packages are integrated environments with point-click-drag-drop graphical interfaces and are not really general-purpose and do not directly support object oriented simulation at an early stage.

Recent commercial object oriented simulation languages such as MODSIM III [7] and later SIMSCRIPT III [8], do not provide or use standard object oriented concepts but use very specific terms and techniques mainly to support backward compatibility with previous simulation software. A comparison study [9] was carried out with PsimJ simulation package and MODSIM III.

5. OBJECTS ANS SIMULATION MODELS

As in Simula, all major components of a system model are identified as *entities*, which have attributes and behavior. Some of these entities will be *active* entities, which have a life of their own. These active entities are called *processes*. The other entities are passive entities, which only exhibit behavior when requested by another entity.

In OOSimL, entities are implemented as objects. An object is an instance of a class, (considered an entity type). A process is an *active object* in the simulation model, and is an instance of a process class. The behavior of an object includes a set of operations (or methods) that are carried out during the lifetime of a software model in execution. An object also includes its own data structures that represent its attributes. In OOSimL, an active object is implemented as a thread object.

Typically, in a software model there are several active objects of the same process definition (class). For example, a model may include several server objects, all with the same definition of the server class. Similarly, there are several passive objects such as the queues to hold customers or other objects.

During a simulation run, all the active objects of the software model interact with each other in some way or another.

6. MANIPULATING PROCESSES

The OOSimL language and associated libraries include facilities and mechanisms necessary to develop simulation models and to carry out simulation runs using the process interaction approach. For constructing simulation models, the language includes the following facilities:

- Definition of processes.
- Starting, delaying, interrupting, suspending, resuming, and terminating processes.
- Starting of a simulation run that will execute for a predetermined period, called the simulation period.
- Definition of various queues that are used in the simulation models.
- Definition of resource pools as passive objects.
- Generation of random numbers, each with its own probability distribution.

7. USING THE OOSIML COMPILER

The OOSimL compiler software consists of an executable program (`oosiml.exe`) and the run-time library, `psimj2.jar`. The general procedure for implementing programs and using the OOSimL compiler consists of the following steps:

1. Compilation of the classes in the OOSimL program
2. Compilation of the Java classes generated by the OOSimL compiler
3. Executing the bute-code program; this effectively performs a simulation run.

The procedure described in normally carried out working with an integrated development environment (IDE), such as jGRASP. The OOSimL Web page includes instructions on how to configure and use the jGRASP IDE to compile and run OOSimL programs. The other way to carry out the procedure is working in a DOS window.

The jGRASP environment is available from Auburn University. The web page is:

http://www.eng.auburn.edu/grasp.

8. A SIMPLE MODEL

In the model of a single-server system, there is only one server object that provides service to the customer objects. An example of a single-server model is shown in Figure 1. The single-server model includes the following components:

- The arrival of customers (cars), indicated with an arrow pointing to the tail of the queue;
- The arrival queue, which is the waiting line for customers;
- The server process, which takes the next customer from the head of the queue and services it;
- The departure of customers, which is depicted by an arrow pointing away from the server.

The Car-wash system consists of a single car-wash machine as the server object. Arriving cars join a line to wait for service. The car at the head of the line is the one that is next serviced by the car-wash machine. After the car wash is completed, the car leaves the system. Figure 2 illustrates dialog boxes for input of parameter values. Figure 3 shows part of the output produced by the simulation run. Two text files are also produced: the trace file and the statistics file.

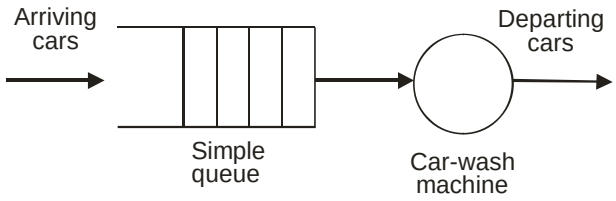

Figure 1. A simple model of a car wash shop.

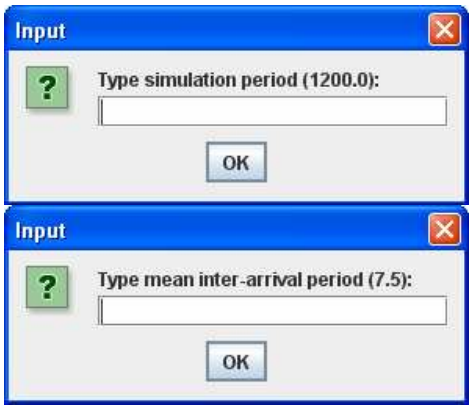

Figure 2. Input data dialog boxes.

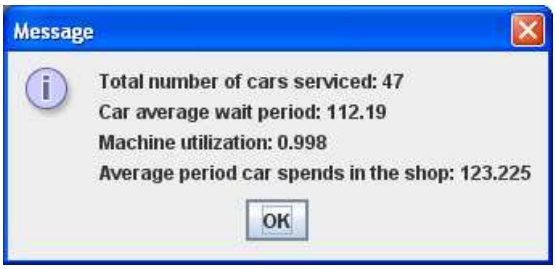

Figure 3. Dialog box with partial results.

It is fairly easy to enhance this model by adding more servers and by defining several types of vehicles, each type with a different priority. For more details on the Car Wash model, the OOSimL simulation language, and the instructions on using the compiler and running simulations, refer to the OOSimL Web page.

The OOSimL compiler, run-time libraries, and related documentation are freely available for educational and research purposes only. The most recent versions of these materials can be downloaded with examples from the OOSimL Web page:

science.kennesaw.edu/~jgarrido/oosiml.html

9. CONCLUSION

The OOSimL simulation language includes all the facilities that were provided by Simula and DEMOS. This makes it possible to implement simulation models with the object oriented and process approaches, which are very powerful for simulating complex systems.

To support the general conceptual framework based on the early introduction to abstraction, object-oriented modeling, logic, discrete-event simulation, and other concepts in problem solving and software development, OOSimL, was developed as a high-level object-oriented programming and simulation language that facilitates the teaching and learning of the programming principles emphasizing object-oriented modeling. The standard pseudo-code syntax and the object oriented features of OOSimL are attractive enough to use the language as a higher level programming language.

10. REFERENCES

[1] Graham M. Birtwistle, Ole-Johan Dahl, Bjoern Myhrhaug, and Kristen Nygaard. *SIMULA Begin* Petrocelli/Charter, New York (1975). ISBN 0-88405-340-7.

[2] W. R. Franta, *The process view of simulation*, North Holland, 1978.

[3] I. Mitrani. *Simulation Techniques for Discrete Event Systems* (with Simula). Cambridge University Press, 1982.

[4] Graham M. Birtwistle. *DEMOS: Discrete Event Modelling on SIMULA*. MacMillan Press London, Basingstoke (1979). ISBN 0-333-32881-8.

[5] Garrido, Jose M. *Object-Oriented Discrete-Event Simulation with Java*. Kluwer/Academic/Plenum Publishers. New York. 2001.

[6] Garrido, Jose M. and Kyungsoo Im. "Teaching Object-Oriented Simulation with PsimJ Simulation Package". *42nd Annual ACM Southeast Conference*. The University of Alabama at Huntsville, Huntsville, Alabama. April 2-3, 2004.

[7] Goble, John. "MODSIM III – A Tutorial". *Proceedings of the 1997 Winter Simulation Conference*. Washington, DC.

[8] Rice, S. V., A. Marjanski, H. M. Markowitz, and S. M. Bailey. "The SIMSCRIPT III Programming Language for Modular Object Oriented Simulation". *Proceedings of the 2005 Winter Simulation Conference*. Washington, DC.

[9] Im, Kyungsoo and Jose M. Garrido. *A Comparison of MODSIM III and PsimJ Simulation Models*. Technical Report May 2004. Department of Computer Science and Information Systems. Kennesaw State University.