AN OPEN-SOURCE VISUAL ENVIRONMENT FOR DISCRETE EVENT SIMULATION: DEOS.

P. Caricato, A. Grieco, F. Nucci, S. Zacchino, A. Anglani

Dip. Ingegneria dell'Innovazione, Università degli Studi di Lecce, via Arnesano, 73100 Lecce, Italy.
{pcaric, grc, nucci, zacchino, anglani}@axpmat.unile.it
http://tsl.unile.it

ABSTRACT

Simulation modeling provides a powerful tool for the study and analysis of actual production systems. Basically, the first step in any simulation process is the formulation of the simulation model. The model should include the description of the main system components. Since simulation is widely used in various application areas, several simulation packages are available on the software market. Each simulation package takes into account different user needs. Currently, the main limitations in simulation software are the limited standard features, difficulty to learn and expensiveness. A still unresolved question is the integration and combined use of different simulation software tools. The present work focuses on these issues supplying a new environment (DEOS - Discrete Event Object-oriented Simulation software) for the development of a simulation package capable to meet most recent user requirements.
A test case, based on the Simlib simulation software and oriented to manufacturing application, has been selected to evaluate the proposed simulation package. Results show the possibility to use DEOS as a high quality certified simulation software to fit modeler requirements.

KEYWORDS

Discrete event simulation, object-oriented software, software quality assurance.

1 INTRODUCTION

System engineering is a key activity for a wide range of markets and industries. Tools are required to help designing solutions, assessing alternatives and, finally, implementing systems by making the best use of existing or future infrastructures.
Simulation modeling supplies a powerful tool for the study and analysis of systems. Computer simulation also provides a fast and inexpensive way for the study of various approaches. This is necessary because of the increasing complexity of modern systems. Therefore simulation is a valuable tool enabling organizations to investigate possible strategies for improving performances and reducing operating costs.
The simulation process first activity consists in the development of the simulation model. This one should include the main system components affecting its behavior.
The success of modeling depends on the correct selection of entities and parameters to be used in the simulation. There is a definite necessity for utilities that assemble the simulation model and code it with no programming effort. Major requirements for such tools (Olufa et al, 1998) are:

1. Allowing users to define the system in terms familiar to them rather than in terms of simulation languages;
2. Checking user's responses for consistency and completeness;

3. Creating new models by the adaptation of existing ones;
4. Providing a visual representation of the system in a form understandable by decision-makers;
5. Avoiding the coding by means of high level simulation constructs at user level.

Simulation is widely used in various application areas, and consequently, there are several simulation packages available on the software market.

User needs have to be taken into account when a simulation software package is developed. User requirements have been evolving significantly in the last few years, (Hlupic, 1999). Currently main limitations in simulation software are: (1) limited standard features/flexibility, (2) difficulty to learn and (3) expensiveness. Whereas, the main positive features are easiness of modeling, good animation/visual facilities, modeling speed and flexibility/linking to external code. Finally, the desired software features are: (1) a better experimentation support, (2) further developments making packages easier to learn/use and (3) better link with other packages. In other words, the available software are predominately easy to use, with good visual facilities, but they are too limited for complex and non-standard problems, too expensive and unable to provide adequate guidance in experimentation.

Many tools for simulation exist, and most of them are available on the market as industrial software products. However, a major problem is related to the integration and combined use of different tools. Indeed, each of them has its own characteristics and capabilities, and can lead to very accurate and useful models of the reality. Nevertheless, very often a given problem can only be studied by using and combining results from several tools, in order to be able to get a satisfactory and comprehensive modeling. Thus, there is an increasing need for an environment in which different simulators can be plugged and exploited in a truly integrated way. Such an environment contributes to the improvement of the system engineering process by enabling a reduction of system design time frames, costs and risks.

An open free environment represents a significant chance to obtain this integration. Moreover, during design and verification, modeling and simulation activities are commonly carried out and should be integrated in the development process to improve:

- reliability: models used in simulation activities can be derived from the design;
- accuracy: the input parameters of a simulation may come from other simulators, or one could use several elementary simulations, each of them devoted to the modeling of a specific function;
- efficiency: integration of simulators within the same environment allows data exchange to occur automatically.

Therefore, from the standpoint of simulation end-users, the integration of models addressing different aspects of system engineering is very relevant and it can be easily obtained in DEOS framework.

The aim of this work consists in developing and validating a complete simulation framework to be tested in the manufacturing area.

In describing a system, it is important to define the components and the way they interact each other. Moreover, it is good manners to declare the valid operations these components engage in and specify how this engagement affects their state before, during and after these operations occur. Then the mapping of these components to the respective terminology in the model environment has to be carried out. From the object-oriented standpoint, the system is composed of interacting physical objects. These objects are typically the central focus of the simulation studies. The main problem is selecting the correct means to model these objects in order to simulate their behavior. The object-oriented approach represents a link between the physical system and the correspondent computer representation. A relevant way of using object-orientation allows the user to develop the simulation model through the selection of construction resources from a simulation library. The logic describing the interaction of these resources is added when user connects these resources together. As a result libraries are developed by simulation programmers and target specific modeling problem.

This paper describes such a simulation package. A simulation library is used to model large scale of system scenarios. The main goal is letting both users exploit it without any simulation programming skill, and programmers extend easily the whole package. Libraries can be created for different simulation environments. In this way, the development of specialized operation libraries in an open environment – such as DEOS – can significantly reduce the complexity involved in developing and implementin g the model.

2 SOFTWARE QUALITY ASSURANCE

Traditionally, the design process of a complex system relied on a (physical) prototype development and long set-up phases. The continuous technological innovation, the market evolution and the competitiveness of today industrial environment require the system design activities to be highly reliable, fast and cost effective. In order to reach such objectives it is necessary to validate the technical solutions and the design choices as soon as possible during the early system architecture design phases. Thus, in order to compete effectively on the worldwide market it is necessary for the industry to obtain an advanced system engineering environment enabling:

- collaborative engineering activities, in order to manage in a concurrent way the design of the different parts (subsystems) composing the system;
- design verification activities, in order to assess the system feasibility and its performances.
- In any industrial sector in which system-engineering teams manage complex systems involving different technical domains, an advanced environment can solve the above problems by providing system engineers with:
- specific tools allowing technical teams to achieve their design tasks such as architecture design, service availability analysis, cost evaluation, mission planning and control;
- modeling and simulation capabilities: the environment should support simulation activities allowing the designers to assess system performance, dependability and behavioral aspects.

One of the expected end-results of the project is a simulation-oriented system engineering environment supporting:

- system architecture design,
- info-model definition,
- simulation configuration.

The use of an advanced environment will strongly improve the system engineering process by providing a high level software platform to allow efficient interoperability of various tools such as simulation tools, estimation tools, impact evaluation tools.

In order to be really inter-operable, the emerging generation of new IT based products and systems has to share conceptual data models, formats and communication protocols. For these reasons, much effort is ongoing in the direction of providing reference info-models in different application domains.

There is no standardized way to present input or output data of simulations to simplify the exchange between tools. The only known attempts to cope with integrated simulations are the DIS and HLA standards (IEEE, 1995), (Dahmann, 1997). Some standards such as STEP (STEP, 1994) exist for data exchange, but they do not allow one to build repositories where data outputs or inputs of simulations may be stored and retrieved.

With the general use of software in industry, system manufacturers are facing a double challenge. On one hand they must develop systems by saving costs and reducing time-to-market. On the other hand they must improve the quality of their systems because malfunctions may cause negative impacts and/or failure of the organization's primary activities. Improved competitiveness can only be achieved through efficient and high quality technologies based on unambiguous notations and methods.

Object-oriented specification, design notations and the associated methodologies are now used almost everywhere when developing new applications. Among them UML (Booch, 1997) is becoming a de facto industrial standard and is currently supported by several commercial available CASE tools, for instance Rose from Rational. One of the reasons of UML success is that it allows one to integrate a system description with parts written in different, possibly formal notations.

In order to provide the single application domain experts with notations and tools tailored to their specific needs some work has been done to build what is known as customizable environments. Such tools, once customized provide the users with an ad hoc notation, which refers to the peculiarities of their application domain. Examples of such customizable environments are MetaEdit+ (Kelly, 1996), ObjectMaker (Mark V System, 1994), ToolBuilder (Lincoln, 1997) and Software through Pictures (Interactive, 1994).

The system engineering market has been dominated by two kinds of actors: (a) the technical systems providers and (b) the end-users of such systems. The current technical approach is dominated by a system engineering view in which different systems, applications and services are integrated. This

integration approach is making the planning, engineering and operation of the various technical systems and services more and more complex for both technology providers and end-users. This is one of the main reasons why engineering methodologies and simulation systems are becoming more and more important. In particular, tools are required to help designing solutions, assessing alternatives and finally implementing systems by making the best use of existing or future infrastructures.
Thus, integration is, on one hand, a clear end-user requirement and, on the other hand, an operational issue for which the technical facilities of today's off-the-shelf tools provide very poor support. Lack of standardization at both model and software levels makes the process of sharing data and information among the models rather complex, costly and sometimes impossible. This is a major limiting factor for a wider and more profitable use of simulation models in system engineering applications.

3 DEOS DESCRIPTION

DEOS has been implemented in order to supply a C++ class hierarchy able to provide a tangible support for the development of discrete event simulation models. The Object-Oriented paradigm has been adopted.
DEOS libraries are based on well-known simulation theory basic concepts: resources, entities and events. The interactions between these elements are managed by a timeline.
A resource describes an element of the simulated system dedicated to perform one or more specific tasks. Typical examples of such elements in the manufacturing field are: machining centers, workers, AGVs conveyors. DEOS resources are black boxes working on entities and collecting information about activities being performed. These data can be analyzed in several ways by end-users or can be sent to other analysis or simulation tools.
An entity represents an element of the simulation environment requiring one or more services. Examples of entities in manufacturing context are workpieces and raw materials.
Events describe actions that may occur in the environment; events may be triggered by a variation occurring to the system or by a change in the system state caused by the occurrence of a condition. Typical events may be the end of a task performed by a resource or the arrival of an entity into the system.
The timeline manages the syncronization of all events that happen within the simulation environment. Every resource and entity is connected with the timeline and schedules events through it. The handling of timeline consists in managing a priority queue containing scheduled events. The basic attribute for the extraction of a generic element from the queue is the occurring time. Fundamental classes for the timeline handling are reported in Figure 1b.
DEOS architecture is implemented through a C++ class hierarchy that gives a representation of the concepts above reported:

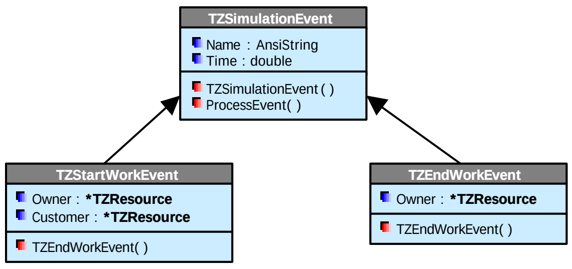
- SimulationObject class represents a generic object that can be modeled in a simulation layout: both Entity class and Resource class are descendant of SimulationObject class (see Figure 1c).
- SimulationEvent class is implemented to give a representation of all the possible events occurring during a simulation (see Figure 1a);
- Timeline class manages a priority queue in which all events are stored and handled; this class allows a correct execution of the simulation.

Class hierarchy is also made of support classes in order to manage special issues, such as simulation errors, statistical distribution properties, collection of simulation output, etc.
DEOS architecture can be extended allowing:

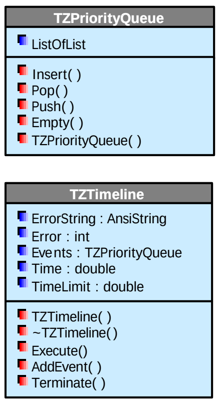
- creation of descendants of resource and entity classes to meet problem requirements, to obtain simulation output in a particular format or to run simulation from inputs given by an external source;
- creation of higher layers using services of the DEOS libraries and allowing to meet special end-user requirements – such as user-friendly graphical interface, fast creation of simulation layout in visual environments, output analyzing tools and animated simulations;

- connection of DEOS to remote software tools by means of special distributed computing protocol, etc.

a) Event classes



b) Timeline classes



c) SimulationObject class and its main descendants

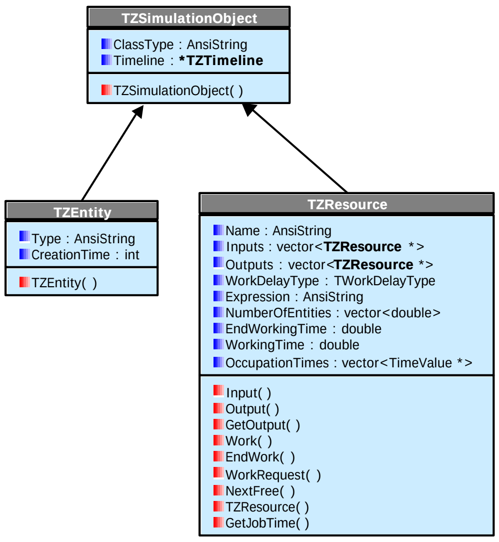


Figure 1: Fundamental classes in DEOS

Figure 2 shows an example of a visual environment, built on top of DEOS libraries in which is possible to draw simulation layouts, setting parameters of each resource and watch simulation result graphical mode. A basic set of classes has been provided for building a variety of simulation models. Instances of these classes can be used in a simulation model by dragging the correspondent graphical elements on the desktop, linking them each other and setting their properties. Additional custom resource classes can be used by: (1) creating the new class as a descendant of the Resource class, (2) implementing the management of the attributes during the simulation running, the statistical information concerning a simulation run, (3) linking the class with the special ones mentioned above.

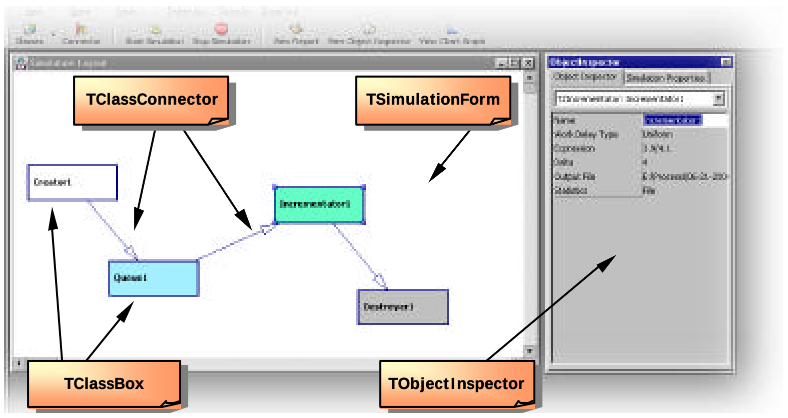


Figure 2: DEOS's visual framework

4 THE COMPARISON CASE STUDY

In order to both check and evaluate the proposed simulation package, a well-known test case has been adopted. In (Oluufa, 1998), simlib software is used to develop a simulation model of a manufacturing system. Moreover, the same model is given in (Schriber, 1991) and (Law, 2000). The aim of the simulation model consists in identifying bottleneck in a production process.
The manufacturing system consists in five work stations. Jobs arrive at the system with interarrival times that are IID exponential random variables. There are three types of jobs, and arriving jobs have different probabilities. Job types require particular tasks to be done and each task must be done at a specified station in a prescribed order. If a job arrives at a specified station and finds all machines in that station already busy, the job joins a single FIFO queue at that station. The time to perform a task at a particular machine is an independent random variable whose mean depends on the job type and the station to which the machine belongs. Assuming no loss of continuity between successive days' operation of the system, 365 eight-hour days have been simulated and the expected overall average of job total delay has been estimated.
From the designing point of view, events in the simlib architecture are divided into three classes: arrival of a job to the system, departure of a job from the system and end of the simulation.
Moreover, a special type variable structure is used in the simlib model to collect information on the average delay in queue for each station and for each job type. Thus, each delay is registered into two different variables, one for the station and another for the job type. For the continuos-time statistics special variables are used, so multiple variables are used for this purpose. In general no possibility exists to implement high quality software in the way described before.

5 RESULTS

A deep analysis has been carried out in order to highlight the advantages of the proposed approach. For this purpose, each single software quality aspect (ISO9000 and ISO9126) has been considered and multiple issues have been reported in the following.
Functionality. Questions related to the integration and combined use of the different software tools can be resolved by adding classes for the interface layer. This aspect is very important because it allows using special tool for specific activities. For example, it is possible to adopt an in-depth data analysis software for the result evaluation or a scheduler package to optimize the simulated system. Moreover, the C++ standard enables programmers to enhance basic DEOS characteristics writing the necessary code.
Reliability. The capacity to behave properly in a specified environment is guaranteed by the C++ compiler features.
Efficiency. DEOS is an extendible set of basic classes that can be compiled under different environment, even though it is particularly designed for C++Builder under Windows®. Nevertheless new generation visual compilers are under way for Linux operating systems. Synchronization question – allowing distributed simulations to run on several workstations or computers – are demanded to operating system characteristics. Indeed DEOS paradigm supposes an executable file is produced after compiling and it has to be launched to perform simulation.
Portability. Portability is particularly considered in DEOS environment because of its OpenSource nature. This allows DEOS based simulation models to run on different architecture by changing the interface layer.
Maintenance. The Object-Oriented C++ environment allows a high maintenance level because of the huge availability of C++ visual compilers. Functions can be easily addressed and modified. The step-by-step running control enhances the programming error tracking. Moreover, classes themselves can be designed to help the tracing activity.
Usability. The graphical framework is a very powerful tool in order to provide support for both the model building and the running phases. Window-based interface provides a quick insertion of simulation model data.

The use of both DEOS and simlib approaches made possible to highlight the need for quality certified software in simulation. Provided that simulation results match, DEOS approach fits better the new requirements in modern simulation world. Complex simulation model runs can be carried out and professional extra tools can be adopted.

6 EXPANDING DEOS FEATURES

DEOS libraries have been developed taking into account the possibility of expanding the basic set of classes and allowing to write extensions for the whole environment. System engineering market often requires flexible tools to meet certain problem specifications. Monolithic software tools – not including expandability – have no chance to cover such requirements. DEOS architecture has been implemented in order to:

- to give developers a flexible set of classes that can be adapted to the problem in just few steps;
- to be a starting point to implement high level layers providing user-friendly graphical interfaces to build, modify and analyze simulation models;
- to supply developers the possibility of adopting modern programming languages constructs, simple interfaces to database and remote servers, networking protocols, distributed computing protocols and infrastructures.

Examples of DEOS extensions are:

- visual interface classes built on the top of DEOS libraries for creating simulation layouts in a graphical way, using all the features offered by recent operating systems;
- an animation layer supplying a real time animation for the simulation model;
- an encapsulation of the DEOS environment in an ActiveX object that can co-operate with other tools of analysis and simulation.

Expandability of DEOS features makes the software architecture capable to meet:

- end-user recent requirements, in terms of user-friendliness, flexibility and interoperability with other tools;
- developer requirements, in terms of code maintenance, extension, reusability and learning.

In Figure 3 a layer view of these extensions is given.

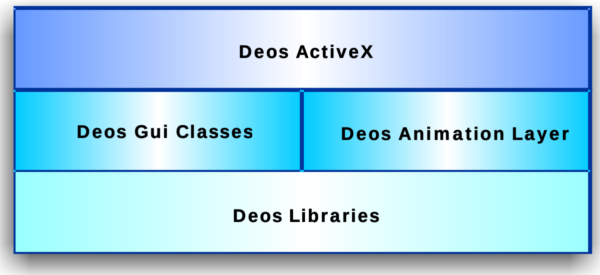


Figure 3: Examples of extensions of DEOS libraries

Expanding DEOS resource and entities libraries is straightforward. Figure 1.c illustrates two kinds of classes that have been designed to represent entities and resources. Developers can write a descendant of these classes to meet special problem requirements, to obtain and store data in different ways or to make a friendly and manageable simulation layout. Entity class descendants can keep tracks of all the working processes acted by resources or can store data taken from a DBMS. A resource class can be written to simulate complex jobs, to create entities from external inputs, such as database servers, or to collect information about its working process in a specific format. Besides, if the simulation model needs more accuracy in analyzing resources behavior, new event types can be defined and added to the model.

7 CONCLUSIONS

In the framework of the current industrial and market context, the main impact of the proposed approach consists in the design and development of a general, standardized software platform for designing and analyzing simulation models. It is envisaged that this will lead to a number of important benefits for both end-users and industrial system vendors in both application domains.
Such users will benefit from DEOS platform by:

- achieving full and efficient interoperability of simulation models that will lead to increased productivity with the consequential cost savings;
- performing integrated simulations that will have a positive impact on commercial costs of simulators, due to the increased availability of the models and the resulting effect in terms of market competition for system vendors;
- allowing interoperability of simulation tools that will decrease dependency of the users from single simulator vendors;
- having a standardized platform, together with a common system engineering data and model repository, that will allow one to try and select different tools for different problems.

DEOS can improve the system engineering process by providing an open platform in which requirements from both end users and developers are satisfied.

ACKNOWLEDGEMENTS

The research has been partially funded by MIUR, PRIN 2000 "Modelli per la pianificazione della capacità produttiva in sistemi di produzione innovativi" and project (488/92) "Piano di potenziamento della Rete di Ricerca e di Sviluppo del Territorio -LINK1" of the Scuola Superiore Sant'Anna, Pisa, Italy.

REFERENCES

Oloufa, A., Ikeda, M., Nguyem, T. (1998), Resource-based simulation libraries for construction, *Automation in construction,* No. 7, (pp. 315-326).
Hlupic, V. (1999), Simulation software: users' requirements, *Computers & industrial Engineering,* No. 37, (pp. 185-188).
IEEE Standard for Distributed Interactive Simulation -Applications Protocols IEEE Std. 1278.1, May 1995
Dahmann J. (1997), High Level Architecture - Overview and Rules, Defense Modeling and Simulation Office, February 1997, URL http://www.dmso.mil /wrkgrps/amg/meetings/siw1/hla_ovr.pdf
STEP, Industrial automation systems and integration - Product data representation and exchange, *International Organization for Standardization*, 1994
Booch, G., Jacobson, I., Rumbaugh, J. (1997), The Unified Modelling Language, version 1.1, *Rational Software Corporation,* September 1997, Available at: http://www.rational.com
Kelly S., Lyytinen K., Rossi M. (1996), MetaEdit+ Afully Configurable Multi-User and Multi-Tool CASE and CAME Environment, *Proceedings of CaiSE '96*, Lecture Notes On Computer Science no. 1080,
Mark V Systems, *ObjectMaker User's Guide,* version 3, 1994
Lincoln Software Limited, What is a meta-tool?, http://www.ipsys.com/white7.htm
Interactive Development Environments, Structure Environment: Using the StP/SE Editors, release 5, 1994
Schriber, T. (1991), *An introduction to simulation using GPSS/H,* John Wiley, New York.
Law, A., Kelton, W. (2000), *Simulation Modeling and Analysis –Third edition,* McGraw-Hill International series.