JDESK WEB-BASED DISCRETE EVENT SIMULATION KERNEL[1]

Inmaculada García, Ramón Mollá, Javier Cabanillas
Computer Graphics Section
Dep. of Computation and Computer Systems
Technical University of Valencia
Camino de Vera, s/n
46022 – Valencia – Spain
{ingarcia,rmolla}@dsic.upv.es
jacatos@inf.upv.es

Abstract

Web-based simulation is a powerful skill for e-learning. Simulation gives concrete answers to the students questions. Having a Virtual Simulation Laboratory is possible when using web-based simulation. This paper presents JDESK, a web-based discrete event simulator written in Java. It is fast, versatile and powerful enough to manage whatever system. This simulator allows to describe and to change a model easily. JDESK may be used as a prototyper. Although it allows simulating any system behavior, it includes libraries containing predefined behaviors to allow students define simple models easily. This simulator allows changing dynamically system topology. Any continuous or discrete behavior may be included in the system. JDESK can manage models with high complexity and size. The programmer only focuses on the model description, not in events or programming bugs. The effort to describe a behavior can be reused (in the same model or others). JDESK includes an assistant to guide the user in the model definition process.

Keywords

Discrete events, web-based simulation, Java, e-learning.

1. INTRODUCTION

Simulation-based e-learning is defined as "*learning by doing*" [Kindley02]. In traditional e-learning, the students are restricted to a prearranged learning path, but using simulation supposes the students select experiences to answer their own questions. The web affects the way simulation is done [Fishwick96]. For educational purposes, the main points are:

- Web-based simulators are accessible from any computer using a web browser. That allows a computer laboratory to become into a Virtual Simulation Laboratory.
- Simulator code is always updated to the latest version.
- The use of multimedia encourage distance learning more so than typical simulation textbook. Such capacity allows more extensive use of resource-intensive media such as audio and video.
- There is not an explicit limitation in the amount of memory for storage of programs, models or pieces of the model previously defined.
- Using models repositories make the code available to be reused and shared.
- Some parts of the simulation are executed on the server and other parts are executed on the local machine, where the web browser is located. That depends on the kind of web-based simulator.
- It is possible to construct simulations where multiple users interact with each other (Distributed Interactive Simulation [DIS]).

1 This work has been partially funded by the Generalitat Valenciana OCYT CTIDIB/2002/344.

A number of web-based discrete event simulation toolkits [Fishman78] have been developed. Some examples are: JSIM, SILK or SIMJAVA. JSIM [Jsim97] [Jsim00] is a Java-based simulation and animation environment. A model is constructed by building a graph with different types of nodes connected by a single edge type. The design diagram may be animated when the simulation is run. JSIM is a component-based software available in internet [Jsim]. SILK [Silk98] is a general-purpose simulation language based on Java. SILK is not so much a language in itself as it is an extension of Java. In SILK, models are developed directly in Java, using a package of classes. The main SILK characteristic is that it allows programmers to develop domain specific objects using JavaBeans methodology. SILK allows creating models graphically, assembling components to quickly create simulation applications. SILK support textual model description. SIMJAVA [Simjava98] is based on the discrete event simulation kernel SIM++. SIMJAVA includes facilities for representing simulation objects as animated icons on screen. SIMJAVA simulations may be incorporated as live diagrams into web documents. A SIMJAVA simulation is a collection of entities each running in its own thread. These entities are connected together by ports and can communicate with each other by sending and receiving event objects. SIMJAVA support textual model description. The software package is freely available in full source code from the home page [Simjava]. In [Kuljis00] discrete event web-based simulators and their applications are discussed more deeply and it makes a comparison between web-based simulation and classical simulation. Those simulators have different ways of model description and operation. Some of those simulators allow sharing knowledge between users, using databases. Some simulators allows to create the model easily, may be in a graphic way. Other ones are text-based simulators, allowing integrating in the simulation other language facilities. But, those simulators do not allow simulating whatever model easily. The simulator restricts system behavior.

All these web-based simulators are written in Java. Some web-based simulators have been developed creating a new language using Java. Other ones port an existing simulation language and create it in Java. Java is an elegant and powerful language with several built-in capabilities that make it quite useful for developing simulations:

- Java is an Object Oriented (OO) programming language. That capability allows creating flexible, modular and reusable programs.
- The simulation models become easily available on the Web. The main purpose of writing simulation toolkits in Java is export simulation code without install the simulation toolkit. It is possible to create a simulation by clicking a web link [McNab96].
- Support to distribute processing.
- Platform independence.
- Multithreaded programming. The multithreaded concept corresponds better with the way the real world system is simulated [Kilgore98].

2. OBJECTIVES

JDESK [Jdesk] is a web-based discrete event simulator written in Java. JDESK must be:

- A general purpose simulator.
- Powerful enough to manage whatever system.
- Flexible enough to allow fanciful behaviors.
- No restrictions to the simulation model.
- Support to real-time simulation.
- Support to external functions.
- Easy model definition and implementation. The system model definition must be done defining the simulation components and their characteristics. The model definition must be easy enough to use it for the learning process
- Easy debugging and changing.
- Easy to learn model definition syntax.
- Implemented as a Java library to allow using in the simulation other Java libraries.
- The simulation file must run locally.
- Platform independence.

In the next sections, JDESK model definition and its working model will be explained in detail.

3. JDESK ORIGINS

JDESK has been based on DESK [Garcia00]. DESK is a C++ object oriented Discrete Event Simulation Kernel API made from scratch. DESK is based on two discrete event simulators: SMPL [Smpl] and QNAP [Qnap]. These simulators are quite different. DESK tries to join the advantages of both, avoiding the disadvantages. The table 1 shows the advantages and disadvantages of both simulators.

CHARACTERISTICS	SMPL	QNAP
Implementation language	C library	Define a language
Model definition	Events and relations definition	Service station and their interconnection definition
Model creation and modification	Difficult	Easy
Model limitations	Number of clients, service stations,	No limitations
Utility	Model implementation	Model validation
Speed	Fast	4 o 6 times slower than SMPL
Model prototyping	Slow	Fast
Model debugging	Difficult	Easy

Table 1. SMPL vs. QNAP

SMPL is a fast simulator, but it is difficult to define the model because that supposes define all possible events in system, scheduling algorithms, time control,… Redefine the model supposes to follow the sequence of events and changing it as necessary. Build complex models can be quite difficult. SMPL is not appropriated to use it as a prototyper, but it is possible to use it to obtain final results, because it is a fast and accurate simulator. Defining a model using QNAP is easy and fast. That only supposes to define all the Service Stations in the model and their interconnection. Scheduling algorithms are predefined in the simulator and time control is done automatically. So, define, debug and change the model is easy. But, the main disadvantage is the low speed of simulation. DESK has been developed following the way of model definition of QNAP, but trying to maintain the simulation speed of SMPL. It has joined the speed and freedom of old C based packages like SMPL and the organization and easiness of an OO model description like QNAP (fast and easy prototyping). In general, DESK uses to be faster than SMPL for almost all simulated models in the laboratory. It is easier to implement or modify models and it offers possibilities that are impossible with others simulators. DESK allows simulation models that are impossible to simulate using SMPL due to the amount of clients in the system, the amount of stations or the model complexity. In general, the cost of developing, debugging and change a model in DESK is much lower than in SMPL.

4. WEB-BASED DISCRETE EVENT SIMULATION KERNEL

JDESK is a kernel developed using Java. The class and expression syntax in Java is quite similar to C++ [McNab96]. Simulation models made by JDESK can easily be implemented and what is more important, easily and securely modified. Model description development and validation is short in time. A top-down or bottom-up strategy can be accomplished to describe model. The programmer only focuses on the model description, not in events or programming bugs. There are many behaviors or random distributions that are predefined and offered by the simulator. So, new users or students can implement simple models very easily. Simulation control can be done in whatever part of the model, for whatever client, station, in whatever model, change schedule algorithms in the stations to exotic behaviors, sophisticated client routing algorithms, very accurate and special accounting,... although there is an accounting facility that shows the basic parameters of all the objects in the system wherever it is demanded. Special features like resources and semaphores are also implemented in the kernel. The simulation model can be programmed in order to detect a critical situation and then, destroy clients, stations, create new ones, change policies, schedule algorithms, client routing, extract results whenever the programmer likes, introduce new stations, delete old ones,... that is to say, the system can dynamically change its topology and structure during the runtime. JDESK is a free web-based simulator, but JDESK libraries are not open source code.

4.1 JDESK Applications

JDESK can be used to solve any problem that could be modeled as a discrete event system [Law82]. Some possible application areas are [Coss92]: manufacturing applications, semiconductor manufacturing, construction engineering, military applications, logistics, transportation and distribution applications, business process simulation, human systems,… Simulation helps to understand the behavior of a system. Using simulation it is possible to change the system characteristics without change the real system.

4.2 JDESK Implementation

4.2.1 Simulator Structures

There are two basic simulator entities: Clients and Service Stations (SS). The user defines the simulation model according to these entities. Clients are objects that travel through the SS, changing their state. Clients contain information that can be modified by the simulator queues and the SS. Clients have associated an identifier, a priority and a class (it serves to classify clients). Clients can change their characteristics dynamically. The key simulation concept of event is in JDESK the change of state or behavior of a client in system. Events management is transparent to user. SS are the simulator dynamic entities. A SS is defined according to its kind, number of servers or resources, service time demanded by clients, scheduling and routing of outgoing clients algorithms, accounting information,... SS contains client queues waiting for service and receiving service. A simulation model is defined by providing the amount of SS in the model, their properties and how they are interconnected. SS can change their specific behavior depending on the characteristics of every client.

Other key structures in the simulator are the clients pool and the events queue (these structures are not accessible to the user). The events queue is composed of clients (the clients model the events that are going to happen in system). Leaving of a client from the SS where it is receiving service it is the only possible event. So, the only possible events queue function is to call the appropriated SS to take out clients from execution at the right time. The events queue is time ordered. Every time a client comes into the system, a Dynamic Memory Manager (DMM) call is performed to reserve memory for the object. When it leaves the system, a symmetric call is done to release the memory space occupied. This situation produces an unnecessary penalty in time performance, because the DMM is constantly being called. In order to minimize the DMM calls, the solution provided by JDESK reuses the clients that leave the system when a new one comes into the system. The outgoing client is not destroyed and therefore the memory is not released like in an orthodox academic programming solution. In that implementation, the client comes into to the pool. When a new client is required in the model and the pool is not empty, a client is unlinked from the pool and it is inserted into the system. Only when the pool is empty, a new client must be created by calling the DMM. Clients are never destroyed during the simulation though they are not in the simulation model.

The SS behavior is defined by parameters and functions (class methods). Predefined classes are provided to create models quickly, but the user can implement its own functions. These functions take charge of managing the service time or the clients creation (sources), inserting clients in the SS waiting queue if it cannot be served at that moment, carrying out the transit of the client after leaving the current SS, number of SS resources reserved by the actual client and special features (the user function is executed when the servers or resources are assigned to a client. The user function allows modeling other not predicted behaviors). Although these functions are defined to carry out a particular task in the simulator, they can be used for any other additional task, giving to the simulator additional flexibility. These functions allow JDESK to change its characteristics dynamically, even during simulation time: creating or destroying SS, changing SS and clients behavior or system topology. They also allow simulator support atypical behaviors: include security alarms, voice recognition, multimedia,... or any another behavior that the programmer considers adequate. Those behaviors are not included in the JDESK kernel.

4.2.2 System Dynamic

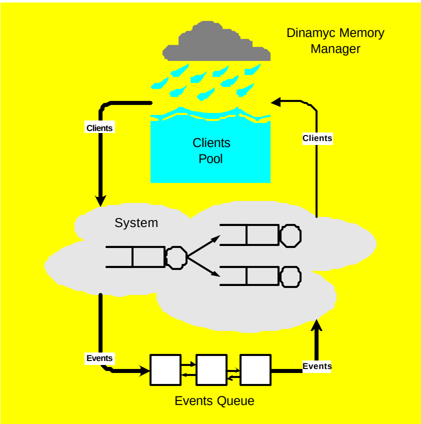

Figure 1: System Dynamic

System dynamic (figure 1) is shared by SS and the Events Queue. To start the simulation a client must come into a SS. If the SS has a free server, the client starts to receive service, generating an event. But, if all servers in the SS are occupied, the client must wait in queue until required. There is only a possible event: a client must leave the SS where it was receiving service because its service time has been completed. When an event happens, the client receiving service leaves the output SS and comes into an input SS (or leaves the system), where the client can receive service (generating an event) or wait for service in queue. Another client can be executed in the outgoing SS, generating another event. This transition produces a free server in the former output station, so another waiting client can be served (generating an event). Events activate the SS. The SS only starts working because of an event or the arrival of a client to the station. Features like client routing between stations, clients generation, the clients I/O from system,... are performed by the SS. The simulator kernel goes through the events queue, popping the head, executing transitions and producing new events until the queue is empty. The simulation finish when the predefined user execution time is reached, the user explicitly finishes the simulation by a given condition or the events queue is empty.

4.3 Model Definition

The main JDESK syntax characteristic is implementation easiness. There are predefined classes or provided by the simulation engineers. This characteristic makes the simulation model code easy to debug, modify and understand. The simulation model is specified by using library functions and is versatile enough to allow the programmer implement whatever desired model behavior. This feature allows learners and beginners to simulate simple models easily at the beginning, providing that all the functionality can be changed and personalized by the user.

JDESK is a text-based simulator. The simulation model is defining as a main Java program. The use of Java-based simulators has advantages for the learning process:

- Using Java-based simulators allows students write simulation code, if necessary, in Java. Each simulator defines its own functions, but the calls to these functions are included in a program written in Java, making easier to write simulation code. Student must not learn a specific language for each simulator.
- The student merges the simulator functions with other Java code.
- The integration of simulation with other applications can be done creating libraries added to the existing simulation classes.

A model is defined by all SS in it and the model topology. The model topology is a SS characteristic; that is to say, each SS contain a behavior function that indicates the destiny SS of a client when it leaves the current SS. The SS behavior is defined by parameters and functions (routing, service time, schedules,...). Some parameters have been defined as constants to allow reading the program easier. No events are defined. Using functions to define the model gives power to the simulator and it allows no explicit restrictions in the model. There are some typical predefined most often used behaviors provided by the kernel, though the programmer can personalize whatever feature. The model is defined using a Java library. The JDESK library classes allows to control the simulation, SS management, clients management, resources management, obtain and select simulation results,…

Steps in the system model definition are:

- Init the simulator.
- Define all SS in system: implement SS behavior functions or use predefined functions and customize SS, behavior and existence.
- Create clients (if necessary).
- Start simulation.
- Obtain results.

Figure 2 models a typical computer closed system, with 4 SS: an infinite number of processors (users), 3 CPU. To receive service in the CPU3 it is necessary to have the BUS resource. The figure 3 shows the source code implemented by the user to define the system. All CPU have the following characteristics: only one server, FIFO non-preemptive scheduling algorithm, exponential service time and the user function is not defined. There are 10 clients in system. The example has been included to show how JDESK works and it shows the implementation simplicity and easiness of modification.

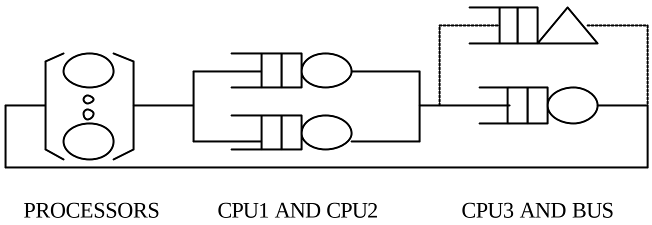

Figure 2. Typical closed system

```
import JDESK.*;
public class Central_Server extends Global {
    JDESK_STATION processors, cpu1, cpu2, cpu3, bus; JDESK_Interface f;
    public Central_Server  () {
        processors = new JDESK_STATION();  cpu1 = new JDESK_STATION(); cpu2 = new JDESK_STATION();  cpu3 =new JDESK_STATION();
        bus = new JDESK_STATION(); f = new JDESK_Interface();}
    // Service time functions
    class t_processors implements JDESK_T_ptrfuncserv { rand r; public t_processors (){r=new rand();}
        public double ptrfuncserv (JDESK_CLIENT c){return (r.expntl(1));} }
    class t_cpu1 implements JDESK_T_ptrfuncserv {  rand r ; public t_cpu1 (){r=new rand();}
        public double ptrfuncserv (JDESK_CLIENT c) {return (r.expntl(2));} }
    class t_cpu2 implements JDESK_T_ptrfuncserv {  rand r;   public t_cpu2 (){r=new rand();}
        public double ptrfuncserv (JDESK_CLIENT c) {return (r.expntl(1));} }
    class t_cpu3 implements JDESK_T_ptrfuncserv {  rand r;   public t_cpu3 (){r=new rand();}
        public double ptrfuncserv (JDESK_CLIENT c) {return (r.expntl(1));} }
    // Routing functions (service station interconnection definition)
    class trans_cpu implements JDESK_T_ptrfunctran {  public trans_cpu (){}
        public JDESK_STATION ptrfunctran (JDESK_CLIENT c) {return bus;} }
    class trans_cpu3 implements JDESK_T_ptrfunctran { rand r; public trans_cpu3 (){r=new rand();}
        public JDESK_STATION ptrfunctran (JDESK_CLIENT c) {
                if (!f.JDESK_Free_Resource (c,bus,new JDESK_T_servers(1)))     {f.JDESK_Error(); System.exit(0); }return processors;} }
    class trans_bus implements JDESK_T_ptrfunctran { public trans_bus (){}
        public JDESK_STATION ptrfunctran (JDESK_CLIENT c) {return cpu3;} }
    class trans_processors implements JDESK_T_ptrfunctran  { rand r; public trans_processors (){r=new rand();}
        public JDESK_STATION ptrfunctran (JDESK_CLIENT c) { if (r.random(1,100)<=30) return cpu1; else return cpu2;}}
    class recurso implements JDESK_T_ptrfuncrec { public recurso(){}
        public long ptrfuncrec (JDESK_CLIENT c) { return 1;} }
    void simul ()
    { int i;
    JDESK_T_schedule plan = new JDESK_T_schedule ();
    f.JDESK_Init();
    processors = f.JDESK_Service_Station("Processors", JDESK_INFINITE,  plan.JDESK_NO_PREEMT, new t_processors(), new JDK _Prior(), null,
                                new trans_processors(), null);
    cpu1 = f.JDESK_Service_Station("Cpu1", 1, plan.JDESK_NO_PREEMT, new  t_cpu1(), new JDESK_Fifo(), null, new trans_cpu(), null);
    cpu2 = f.JDESK_Service_Station("Cpu2", 1, plan.JDESK_NO_PREEMT, new  t_cpu2(), new JDESK_Fifo(), null, new trans_cpu(), null);
    bus   = f.JDESK_Resource_Station("Bus", 4, new  recurso(), new JDK _Fifo(), new trans_bus(), null);
    cpu3 = f.JDESK_Service_Station("Cpu3", 1, plan.JDESK_NO_PREEMT, new t_cpu3(),  new JDESK_Fifo(), null, new trans_cpu3(), null);
    for (i=0; i<10; i++) f.JDESK_Insert_Client (new JDESK_T_client_number(i+1), new JDESK_T_client_prior(JDK _NO_PRIOR),
                                  new JDESK_T_client_class(JDESK_NO_CLASS), processors);
    if (!f.JDESK_Simulation("Central Server M/M/1", new JDESK_T_system_time(0), new JDESK_T_system_time(25)))  f.JDESK_Error();
    else { f.JDESK_Results_Table (processors, true); f.JDESK_Results_Table (cpu1, false); f.JDESK_Results_Table (cpu2, false);
        f.JDESK_Results_Table (cpu3, false);    f.JDESK_Results_Table (bus, false);     } }
    public static void main(String[] args)   {  Central_Serve rp=new Central_Server();    p. simul ();  }
}
```

Figure 3. JDESK example source code.

4.4 JDESK Interface

JDESK is a text-based simulator. The simulation model is a main Java program that uses JDESK library functions. The main program must contain the model definition and the SS behavior functions. The main simulator window contains an editor window, a message window and controls. The simulator allows writing the code in a text editor window. The simulation code can be copied and pasted from other applications to the editor window. The code must contain a main Java program. The model must be defined following the steps previously defined. Previously defined models or examples can be used to construct the model. It is possible to add behavior functions previously defined or predefined JDESK functions. But, this way of model definition it is only recommended for expert users or to modify an existing model. JDESK includes an assistant to guide the user in the model definition process (figure 4). The assistant is a Java Applet, which allows creating models easily, using examples, controls or classes skeletons. It constructs the text-based model using the user specifications. The code written using the assistant can be edited in order to change the model as necessary.

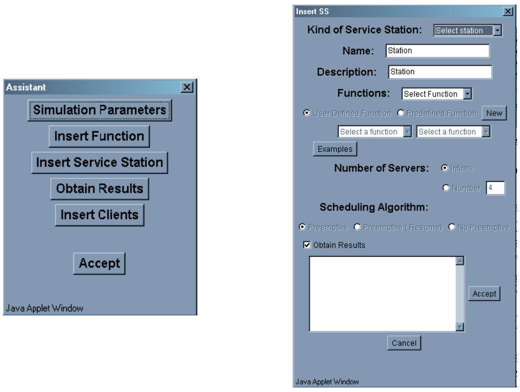

Figure 4. JDESK Assistant.

Using the assistant controls it is possible to define all the SS in the model (servers, sources and resources). When the insert SS button is pressed, the user is asked to define the kind of SS to include and its parameters only by clicking buttons, radio buttons or menus. The user only writes the necessary code, that is, some special SS behavior functions. The Assistant allows:

- Define the general simulation parameters, as the global simulation time.
- Add behavior functions to the simulation model. The assistant show previously defined behavior functions and JDESK predefined functions. The user can select one of these functions or write the function in the editor window. If the user selects to write the function, a function skeleton is shown in order to avoid the user to write the code that is common in all functions and to guide him in the function creation process. That functions are methods of a given class. The assistant fills the class structure when the function is written. So, the user only writes the strictly necessary code.
- Add a SS to the model. The assistant allows selecting the kind of SS. Depending on the SS kind, it shows the controls to define the necessary parameters. Some SS behavior is defined meanwhile class methods. Those functions could be selected between a collection of predefined functions (to model typical SS behaviors). The SS behavior function could be previously written or in the SS definition process.
- In closed systems it is necessary to include clients in the model. The assistant allows creating clients and assigning them to a SS. The client parameters (identifier, priority and class) could be defined using the assistant. It is possible to identify those clients with sequential identifiers starting on a given value.
- Obtain simulation results: the assistant allows selecting the results to show and their format.

Once the model is constructed, JDESK compiles the program using a control and it allows obtaining the Java simulation files. The user may run the simulation locally. The simulation is a set of files that can be ported to different platforms using different Java compilers.

The JDESK compiler is running on a server and a user could access to it using the JDESK HTML page. A JDESK user must have installed in the computer a web browser and a Java Virtual Machine. The web browser must allow to run Java Applets. The steps in the simulation process using JDESK are:

1. Use the web browser to access JDESK HTML main page [Jdesk]. It is necessary that the web browser has the Java option activated. If the local computer has not a Java Virtual Machine installed, it is be automatically downloaded.
2. Write the simulation model in the editor window or using the assistant. The Java Applet runs on the local computer, so, there is not necessary an internet connection during the code edition.
3. Build the simulation model. The build process creates the simulation files in the server, compiles the user model and links it with the JDESK libraries. The compile process result is shown in the message window. In this simulation step it is necessary to be connected to the JDESK web page.
   3.1. If the simulation built fails:
      3.1.1. Make changes in the simulation model as necessary, taking charge of the messages from the message window.
      3.1.2. Go to the step 3.
   3.2. If the simulation built success: go to the step 4.
4. Download the simulation files. JDESK creates a compiled file (.class) for each user class in the simulation model. It is necessary to download all compiled files. JDESK allows downloading the simulation source file too (.java). The source file is not necessary to run the simulation.

5. Runs locally the simulation using any Java Virtual Machine. The simulation results are shown in the standard output.

5. CONCLUSION

JDESK is a web-based simulator written in Java, so, it is possible to access it using a web browser, allowing using whatever computer laboratory as a Virtual Simulation Laboratory. The main characteristic of JDESK is the implementation and modification easiness, so it can be used as a prototyper. This makes JDESK suitable for e-learning. The system model definition is a description of the SS in model and the model topology. The JDESK assistant allows the student define models easily, writing only a few lines of code. The simulator includes predefined behaviors that can be used to define the great part of models. But special behaviors can be modeled too, allowing the student satisfy his learning objectives. JDESK does not impose constraints in the model definition, it allows defining whatever system. The model topology or behavior can change dynamically. The simulation code is implemented in Java, what grants the platform independence.

6. REFERENCES

[Coss92] R. Coos. Simulación: un enfoque práctico. Limusa, Mexico.1992.

[DIS] http://www.sei.cmu.edu/publications/articles/arch-dist-int-sim.html

[Fishman78] G.S. Fishman. Conceptos y Métodos en la Simulación Digital de Eventos Discretos. Limusa, Mexico, 1978.

[Fishwick96] Paul A. Fishwick. Web-Based Simulation: Some Personal Observations. 28th Winter simulation conference, California 1996.

[Garcia00] I. García, R. Mollá, E. Ramos, M. Fernández. D.E.S.K.: Discrete Events Simulation Kernel, ECCOMAS 2000, September 2000

[Jdesk] http://www.sig.upv.es/investigacion/Simulacion/JDESK.htm

[Jsim] http://chief.cs.uga.edu/~jam/jsim/

[Jsim00] J.A. Miller, A.F. Seila, X. Xiang. The JSIM Web-Based Simulation Environment. Future Generation Computer Systems, Volume 17, issue 2 (October 2000). Elsevier Science Publishers B.V. Amsterdam, The Netherlands. Pages: 119 - 133 Periodical-Issue-Article. ISSN: 0167-739X.

[Jsim97] J.A. Miller, R.S. Nair, Z. Zhang, H. Zhao. JSIM: A Java-Based Simulation and Animation Environment, 30th Annual Simulation Symposium (SS '97). April 1997. Atlanta.

[Kilgore98] R.A. Kilgore, K.J. Healy, G.B. Kleindorfer. The future of Java-based simulation. 30th conference on Winter simulation, p.1707-1712, December 1998, Washington, D.C., United States

[Kindley02] R. Kindley. The Power of Simulation-Based e-Learning. The e-Learning Developer’s Journal. September 2002. www.eLearningGuild.com

[Kuljis00] J. Kuljis, R.J. Paul. A Review of Web Based Simulations: Whither we wander?. 2000 Winter Simulation Conference. Baltimore.

[Law82] A.M. Law, W.D. Kelton. Simulation Modeling and Analysis. McGraw-Hill Series in Industrial Engineering and Management Science. 1982.

[McNab96] R. McNab, F.W. Howell. Using Java for Discrete Event Simulation. 12th UK Computer and Telecommunications Performance Engineering Workshop (UKPEW), 1996.

[Qnap] http://www.simulog.fr/formation/qnam.htm

[Silk98] Introduction to Silk and Java-Based Simulation. Kevin J. Healy, Richard A. Kilgore. 30th Winter simulation conference, Washington, DC, United States.1998.

[Simjava] http://www.dcs.ed.ac.uk/home/hase/simjava/

[Simjava98] F. Howell, R. McNab. Simjava: a Discrete Event Simulation Package for Java with Applications in Computer Systems Modeling. First International Conference on Web-based Modeling and Simulation, San Diego CA, Society for Computer Simulation, Jan 1998.

[Smpl] http://www.autoctrl.rug.ac.be/ftp/smpl/