GPSS Interactive Learning Environment

Villarreal Gonzalo L. [1], De Giusti Marisa R.[2], Texier José [3]

[1] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) and Proyecto de Enlace de Bibliotecas (PrEBi UNLP), 49 street and 115, La Plata (1900), Argentina

[2] Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CIC) and Proyecto de Enlace de Bibliotecas (PrEBi UNLP), 49 street and 115, La Plata (1900), Argentina

[3] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Universidad Nacional Experimental del Táchira (UNET) and Proyecto de Enlace de Bibliotecas (PrEBi UNLP), 49 street and 115, La Plata (1900), Argentina

ABSTRACT

This work presents an open source web environment to learn GPSS language in Modeling and Simulation courses. With this environment, students build their models by selecting entities and configuring them instead of programming GPSS codes from scratch. Teachers can also create models so that students can apply, analyze and interpret results. Thus, it includes a simulation engine that stores snapshots of models as they are executed, and allows students to navigate through these snapshots. The environment may be combined with existing learning management systems.

Keywords: *Mathematical modeling; prospective mathematics teacher; teacher instructional practices; case study.*

INTRODUCTION

Programming discrete simulation models can be successfully accomplished using general purpose programming languages, such as Java or C++, or specialized languages like General Purpose Simulation System (GPSS) or Simula. While the former are known by any programmer, the latter introduce entities and processes in a higher abstraction level as well as a set of analysis tools to obtain data from simulated models. Thus, simulation languages greatly facilitate the development and execution of simulations of complex real-world systems. Most general purpose programming languages are designed around a set of instructions that control the execution of an algorithm: sequences, iterations and conditions are organized in objects, routines and subroutines. According to this simplification of programming languages, learning to build programs requires understanding what basic routines do and how they can be combined to deliver a recipe for the computer to execute.

The behavior of a system as it evolves over time is usually studied by developing a simulation *model*. This model usually takes the form of a set of assumptions concerning the operation of the system and, once developed and validated, it can be used to investigate a wide variety of "What if?" questions about the real-world system. Each simulation language generally possesses an orientation to real-world situations, which may be classified as event-oriented or process-oriented. Simulation languages exist to make it easier to build models for analysis and to answer those *what-if* questions. Understanding the output of a simulation after it was executed is as important as programming the model. To learn an event-driven simulation approach, a few major concepts need to be incorporated: entities, events, queue theory, simulation times, etc. Thus, learning a simulation language requires understanding the syntax and semantics of the language to manage those new concepts. Unlike general programming languages, simulation languages are not built around basic or atomic instructions. Instead, they are based on high level sentences representing entities with abstract attributes and specific behavior. The combination of these sentences describes several system components and their interactions. Simulation languages are so powerful because they allow programmers to create complex models with very few lines of code, run those models and finally retrieve information – general data and statistics – that explains the behavior of the system. In the academia, GPSS (General Purpose Simulation System) is a widely used alternative for modeling and simulation used for teaching discrete systems. GPSS is processes-oriented and defines the structure of models based on a set of language commands. Each

command describes processes, with attributes and subroutines inside, which affect the model being represented. The language supplies lots of entities for the programmer and more than 50 commands to deal with them: create, use, release and query their status. There exists a small set of available tools and integrated development environments (IDE) for GPSS dialects. Most of them offer a place to write the model and to run it and they display reports regarding the simulation execution results. Code assistance and visual aids for the user during the programming stage are rarely seen here. Besides, deep analysis tools of how the simulation – and its entities – progressed and changed are quite unusual too. All of these tools are tremendously useful once programmers know GPSS well, but they are not suitable for newcomers that need to learn everything from the ground up. Consequently, students spend most class time learning how to specify and combine entities in a particular language instead of focusing on acquiring important simulation skills, such as mapping real systems into abstract models, identifying input and output variables and so on.

In this work, a web-based interactive learning environment (ILE) for GPSS is presented. The ILE was built upon an intuitive graphic user interface (GUI), based on dialogues and visual aids and a simulation engine improved for learning purposes. As with most web applications, no installation process is involved since the whole application can be exposed through a web server. In addition, the application architecture –split into several layers based on the client-server model– enables the ILE to be plugged into other web systems, in particular Learning Management Systems (LMS) such as Moodle (Moodle, 2012) Sakai Project (Sakai Proyect, 2012) and LRN (LRN, 2012). Models created on the client side of the ILE can be sent to the simulation engine on the server for its execution. The engine handles everything related to code parsing, interpretation, simulation running and report generation, keeping the client lightweight. Last but not least, for each simulation run, the server registers all changes as different simulations' snapshots which are kept in persistent media. Thus, students can fetch *snapshots* of their simulations and analyze in deep detail how the whole model –or even a particular model entity– progressed during execution time. The following section shortly reviews some previous modeling and simulation related publications as well as current GPSS implementations. Next, in the *Methodology* section, some issues related to GPSS learning are explained and the most important aspects of this work are introduced. Then, the application is presented and its implications related to those issues are described, followed by a section with some important notes about the development and its advantages. Finally, suggestions for future work have been included in order to offer a wider point of view of the project.

RELATED WORK

Simulation teaching in general and GPSS learning in particular are topics that have already been discussed over the last decades. Zikic and Radenkovic (Zikic, A. M.; Radenkovic, L. J, 1996) discussed GPSS learning problems. After some years of observation, they have summarized in four points the reasons why it takes so long for students to accept an entirely new approach to programming:

- Students acquired programming skills based in Pascal or C, which employ strict and explicit typing for variables/objects;

- GPSS does not have explicit declaration or strict typing of variables and objects;

- Pascal and C propose a very structured and organized approach for programming;

- GPSS syntax is of assembler type, with labels and transfer statements, which completely clashes with structured programming;

In their work, the authors introduce a new simulation language, based on GPSS concepts but oriented to a structured programming layout. This language, named ISDS, has different syntax rules and semantics compared to traditional GPSS dialects, and introduces some important changes such as explicit declaration, types and routine code organization. Even though the base concept seems correct, the authors suggested that it could delay the learning process in a second stage: the students would eventually have to use GPSS or some other commercially available system. Thus, once they have learned ISDS, they would still need to learn GPSS or another simulation language.

There exist few active implementations of GPSS language, among which MinutmanSoftware's version, named GPSSw (Minuteman Software, 2012), excels other developments as the most widely used one. GPSSw is a desktop application, and quite fast for developing and running simulations under MS Windows. To code models, GPSSw users have at their disposal a plain text area to write GPSS sentences. The application allows the creation of the simulation as well as to start its execution by inserting a *START* command. Simulations are usually run in a few seconds; the results are displayed in a text report which summarizes the most important aspects of the previously run model. GPSSw also offers some debugging tools such as pause and resume points during runtime, and some live examination tools for most GPSS entities (facilities, storages, transaction chains, etc...). GPSSw is also very useful to generate

screening, optimization or user defined experiments. Wolverine Software Corporation (Wolverine Software, 2012) has also developed a GPSS implementation, which shares some common points with GPSSw. It is also a MS Windows application which lets users run models over the simulation engine and outputs a simulation report similar to that of GPSSw. However, models have to be compiled first, which results in a MS Windows executable program representing the simulation ready to be run. Compiling source files allows users to define external so-called ampervariables (routines) in C++ programming language, which will be compiled altogether with the model file(s). Even though the compilation process adds an extra step, it proves very useful for users to enable the extension of the simulation engine, either to connect the model to external data sources or to generate different output files. WebGPSS is another application that implements GPSS dialect. Models are created by defining and connecting block diagrams. This facilitates learning since engineering students are used to representing models with diagrams and charts. Models are created in a client desktop application and sent to a server for execution. The term *web* is owing to the fact that the server process could be run in a remote computer and thus be accessed. Both server and client are executable programs included in the package.

As learning tools, all GPSS dialects presented above share some common aspects that students and teachers need to deal with. As mentioned before, no current GPSS implementation includes either code assistance or syntax highlighting. On one side, WebGPSS implements a drawing-based alternative in which the user chooses blocks from figures and arranges them in a connected network. On the other side, GPSSw includes a plain text editor and GPSS/H has no editor at all. All GPSS alternatives require some installation process and all of them are also MS Windows dependent. Even though all alternatives can be run in Unix-based platforms using an MS Windows layer like Wine or Crossover, there is of course some performance drop. Besides, and more important, although some developers include a limited or trial version, there exist neither open source nor free alternatives available. In education environments, only WebGPSS includes some visual aid for the user. However, all mentioned versions are intended for users that already handle GPSS syntax, semantics and entities usage. In addition, there is no possible integration with existing web LME's.

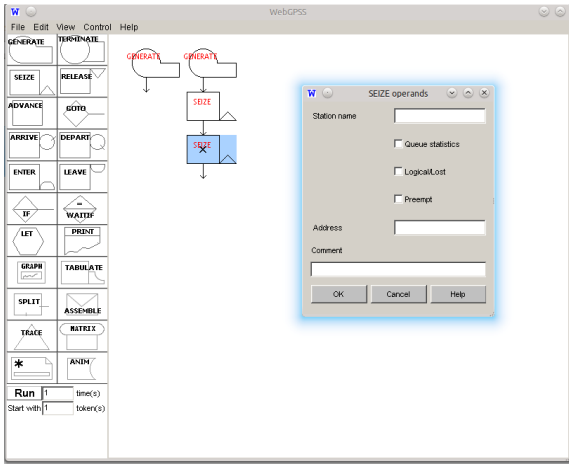

Figure. 1: WebGPSS IDE displays dialogues and graphs

Desporovic et all implemented a GPSS/FON (another less used GPSS dialect) based environment, which they called FONWebGPSS. The purpose of their work is to combine a GPSS implementation with Moodle. With this purpose in mind, teachers prepared case studies and problems related to the area of discrete event simulation, and students used GPSS/FON via Moodle to write and run the models. Simulation execution results are also integrated into Moodle, mixing standard GPSS reports with statistics charts created ad-hoc. Although the simulation engine was not adapted for teaching purposes, this work exposes the need to transform simulation learning via web environments, and to integrate simulation tools in existing LMS. In a different approach, Fonseca et all designed a framework with a Java-based desktop application to build GPSS models from graphics. The framework lacks, on purpose, the implementation of the simulation engine. The main idea of their work is to let students complete the engine by programming themselves the behavior of GPSS entities. This would help students understand how the engine works and would motivate them to write their own simulation engine. In addition to the *empty shell*, JGPSS also includes a regular GPSS engine and statistics generator. Note that in that work the authors considered the importance of understanding a

simulation engine by accessing its core implementation and modifying its source code in a language they feel comfortable with.

METHODOLOGY

Between 2005 and 2011, during the development of Simulation and Modeling courses in Computer Science School at the National University of La Plata, students mainly used Minuteman Software's version of GPSS. During that period, some common obstacles that most students came across when starting with GPSS were identified:

- Unlike general purpose programming languages, GPSS has many sentences and only two sentence categories: blocks and commands
- Each sentence may require a different amount of parameters; some of them are mandatory and some are optional, some refer to other entities and some expect numeric expressions only
- Although GPSS blocks are semantically different from GPSS commands –and that difference is key to understanding the simulation model– they are displayed in the same way, in the same place and they might even be syntactically mixed
- Some blocks are used for general flow control, while others are used specifically to deal with a single entity
- There is no code assistance or highlighting. One of the obstacles that arises from this is that it is hard to remember which entities have already been defined –especially in large models– and how they must be referenced
- When the simulation ends, the system displays a report which resumes in few statistics what happened during the execution of the model. Thus it is not easy to know what really happened: when entities were created or destroyed, how the system clock advanced or which entities were interacting
- The code organization and the language itself are very different from other programming languages that students *had learned before (i.e. C++, Pascal* or Java)

This paper proposes an interactive learning environment (ILE) to improve GPSS learning which tackles all previous issues in the very first stages of simulation courses. The bottom concept of the ILE is quite simple: instead of programming models by writing code from scratch, students can build their models by selecting and configuring entities and sentences. To that end, the GUI has been designed to explain some commonly confusing aspects of GPSS from the beginning and to allow students to focus solely on core simulation concepts and entities. The application presented here is suitable during the simulation development stage and also after the simulation has been executed, in the results analysis stage. In the next two sections, most elements introduced in both stages (development and analysis) are expounded.

Aids During Development Time

During the stage of development, students need to focus on their model and the flux of entities through it. They should not be distracted by issues of one particular programming language. To that aim, a first visual element introduced is the use of dialogues to insert blocks and commands, instead of writing code. Many efforts have been put into dialogue boxes, including a set of small but useful visual elements that avoid dispersion and most common mistakes.

- Dialogue boxes show all possible parameters for each sentence (block or command)
- Parameters include hints for the user to get a clue of what they are for
- Mandatory fields are clearly distinguished from optional data
- Since some sentences require the use of previously created entities, many dialogues display a list of optional entities to choose from
- Listed entities are just the ones having the correct type required for the sentence which narrows the

possibilities to cause a mistake, mixing sentences with entities of incorrect type

In addition to dialogue boxes design, the ILE displays available blocks and commands grouped by their main function or by the entity they deal with. For example, since ENTER and LEAVE blocks are used to access and free *storage* units, they are shown in the same group; similarly, given that SEIZE, RELEASE, PREEMPT and RETURN blocks exist for *facilities* manipulation, they are shown in another group. Grouping blocks helps students make associations between blocks and entities and among related blocks too. Block groups are also separated from command groups, which reinforces the idea that, even though they look similar, they are not. Once confirmed, dialogues create GPSS sentences and entities in an inner representation and they are displayed in a GPSSW-like dialect. Code is rendered highlighting keywords and parameters, and again sentences have been split into two code zones: commands above and blocks below. In addition to GPSS code representation, blocks are presented as a list, and commands as a queue, which is the actual internal representation of the simulation engine. Among other features, blocks and commands can be moved up/down, dragged and dropped from the list (or queue respectively) and removed. Code parts can be intuitively organized, although blocks and commands can never be mixed. The GPSS model, created in the web client application, can be sent to a server for its execution, which contains a GPSS interpreter and a modified simulation engine that runs the simulation.

Simulation execution analysis

GPSS is designed for the sub-area of discrete simulation systems, which basically means that each simulation is controlled by a finite simulation clock that changes discretely. For each clock change (*tick*) the system being simulated updates its status, cascading to all entities in order to be ready for the upcoming clock change. This behavior affects both permanent entities (facilities, storages, chains) and temporary entities (transactions). Hence, a simulation run can be decomposed as a sequence of clock changes and entity updates associated with those changes. Each component of that sequence represents a *snapshot* of the simulation in a precise clock time. This concept leads to a different way of interpreting a simulation run, based on the analysis of entity changes during running time. Even though this might not be very useful for real simulation analysis, it clearly helps students understand how the simulation advanced and what *really* happened in each advance: how the clock moved forward, which entities existed in each *clock time*, what they were like and how they interacted with each other. It is of particular interest that this model also helps to understand the transaction scheduling system inside the system chains, which is a particularly problematic topic. The previous concepts were implemented by making major changes to the GPSS interpreter and the simulation engine to enable them to take full snapshots of the simulation during runtime. Snapshots are taken each time the system clock changes and they are queued in an in-memory data structure as the simulation runs. It must be considered that snapshots are quite heavy in memory and CPU terms, since they represent the whole simulation graph, composed by all model entities and its relationships. Once taken, each snapshot is persisted in a relational database system (RDBMS) for further analysis. Persistence is also a heavy task that involves saving all entities to a relational database and it is handled by a concurrent thread that maps objects to tables and stores them in the DB in disk.

When the simulation is sent to be run in the server, it only takes few seconds until the execution ends but it could actually take a while to be fully stored by the persistence thread. To make that clear for the user, the system GUI has been designed to display a brief simulation report in the final stage and to show the user the persistence process as it advances. Once all snapshots have been persisted, the GUI enables the user to access snapshots, sorted by its system clock. Then, the student can browse snapshots, and inspect the status of self created entities as well as system default entities and data structures. Each entity can also be selected individually, which results in a specific report for the entity itself displaying it status in each system clock and other entities being affected. Again, these entities can also be selected, allowing the user to recursively navigate through entities and simulation snapshots.

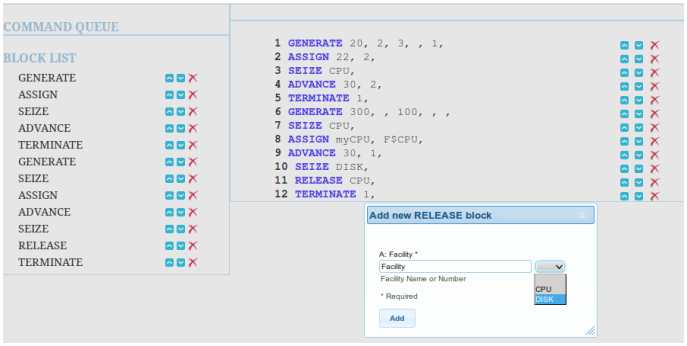

Figure 2: GPSS ILE: dialogues display entities lists and field hints

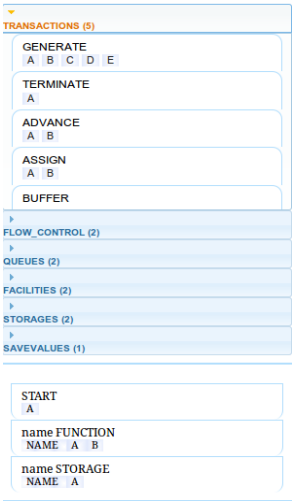

Figure 3: Block and commands grouped by function or entity

RESULTS

The application has been organized into a multi-layer architecture mainly composed by a thin client on one side and GPSS interpreter with the simulation engine and the RDBMS on the other side. Both parts of the application are completely independent one from the other. For them to work together, a MVC application has been developed to acts as a bridge between them, transforming requests from the client to messages to the interpreter and vice versa. The high decoupling level also allow application parts to be easily embedded into other environments. The client side was built using only HTML, Javascript and Cascading Style Sheets (CSS), and works as an independent pluggable module. The GPSS interpreter and engine is a common open source Java and MySQL application. It has been adapted to the Maven2 standard, which makes it really easy to download, extend and compile. Even though it is not likely that GPSS students will try to modify or extend something they still do not understand at first, accessing the source code of the simulation engine is a powerful way to understand how GPSS entities really behave inside, by reading it in a well known language. Similarly, having snapshots that compose a simulation stored in a database gives the students an opportunity to write their own SQL queries and to research much deeper about the entities than any front-end could offer. The application can also load and execute *preexisting models*; teachers can create their own models and leave them available for students to understand particular issues, review any class subjects or prepare quizzes and activities. Once models are created by the teacher, students will see an icon to load them in one click. After a model is loaded, users have full access to it just as if they had written it block by block and command by command. They are able to modify it too or even to extend it, which proves very suitable for some activities of interpretation and system improvement or optimization. In the meantime, as they analyze the model, the system will have sent it to be run in the background and at the end it will output the same kind of results as any model created by hand.

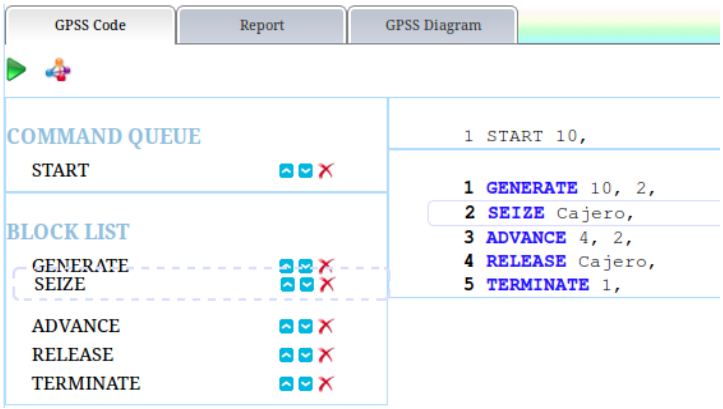

Fig. 4: Drag and Drop over the block list

Social implications in education were also considered in this work. An open source application results in a helpful teaching tool and the ability to plug it into other systems represents a great opportunity for teachers and students. In third world countries, license fees are hard to afford. Commercial applications and trial versions are not a real solution, although they mitigate limitations. Platform dependent applications make it even harder for non Windows users. By using a web application, students can use it in any computer, tablet or even mobile phone: all they need is an operative system with a web browser. No installation process is required. Last but not least, colleges can naturally host their own GPPS full stack: client, web application and interpreter. This allows them to count with a full repository of new learning objects: GPSS models ready to be run, modified or extended, and already executed GPSS models to deeply analyze and understand them from the ground up.

CONCLUSION

A web application for GPSS learning has been presented here. The application is targeted to students in their very first encounter with modeling and simulation subjects, in special with a programming language like GPSS. The main purpose of this work is to shorten simulation learning times by eliminating the need for students to deal with specific issues of a particular simulation language and letting them focus solely on important modeling and simulation concepts such as abstraction, resources, statistics and experimentation. To that end, the application focuses on two key stages of modeling and simulation learning: model building and simulation run. The former is tackled by providing the students with an interactive GUI that helps them build the model by selecting and configuring sentences, organizing code parts, and laying GPSS elements out making key aspects clear from the beginning. The latter is faced under the premise that a simulation run is too complex to be understood from a single text report with some statistics results, especially when students are still learning what a simulation is or should be. The solution presented proposes a *navigable report*, where the students can surf from their simulation entities and access the state of each one in all simulation clock changes. This way, they can understand how entities are affected by other entities and how the simulation system deals with the execution of temporary entities created on the fly. Another important aspect of this work is that it has been released as open source software. This license has two main implications in this context. First, and as with any other open source software, anyone can download, use and extend the applications as needed; of particular interest is the possibility of integrate it with other LME's or with any web application. Secondly, students can be encouraged to study the software as it is, to improve it and extend it, either on their own or as a class exercise.

FUTURE WORK

There is much work to do to improve this environment. Many but not all GPSS entities have been implemented, so a first improvement would be the inclusion of more entities and their associated sentences. Dialogue boxes can also be improved by including visual clues for students to make associations between GPSS entities and real-world objects: if the student is creating a FACILITY, the GUI would show something that *works* as a facility, like a toll barrier or a computer processor. Reports can also be extended to include new entities and to allow students to compare different snapshots side-by-side, highlighting changes in existing entities as well as entities created on one side only. The ILE still lacks a view for the professor, to create models and organize activities and class members. Even though the latter can be accomplished by most existing LMS, model creation, testing and publication must be improved to encourage teachers to add new models for their students. Integration with existing LMS must also become better. As any web application, the GPSS ILE in its current state can be embedded into any web page using, for example, an iframe. However, it would be very helpful to integrate it with existing LMS via installable plug-ins/extensions. This would demand many efforts, since each addition must be customized by hand, but would encourage system administrators to offer the tools to its users.

REFERENCES

Aho, Alfred V et al. Compilers: principles, techniques, and tools. Pearson/Addison Wesley, 2007.

Banks, Jerry, and John S Carson. "Discrete event system simulation." (1984).

Cox, Springer W. "GPSS World: a brief preview." Simulation Conference, 1991. Proceedings. Winter 8 Dec. 1991: 59-61.

Crain, Robert C. "Simulation using GPSS/H." Proceedings of the 29th conference on Winter simulation 1 Dec. 1997: 567-573.

Despotovic, MS, BL Radenkovic, and DM Barac. *GPSS for e-learning environment*. Telecommunication in Modern Satellite, Cable, and Broadcasting Services, 2009. TELSIKS'09. 9th International Conference on 7 Oct. 2009: 318-321.

Fonseca i Casas, Pau, and CJ Casanovas: *JGPSS, an open source GPSS framework to teach simulation*. Winter Simulation Conference (WSC), Proceedings of the 2009 13 Dec. 2009: 256-267Ref 4

Garcia, Heriberto, and Martha A Centeno. "SUCCESSFUL: a framework for designing discrete event simulation courses." Winter Simulation Conference (WSC), Proceedings of the 2009 13 Dec. 2009: 289-298.

Kirkerud, B.: Object-Oriented Programming with SIMULA, Addison-Wesley, 1989

Klein, Ulrich, Steffen Straßburger, and Jürgen Beikirch. "Distributed simulation with JavaGPSS based on the High Level Architecture." SIMULATION SERIES 30 (1998): 85-90.

Kleijnen, J.P.C. 2005. Supply chain simulation tools and techniques: a survey. *International Journal of Physical Distribution & Logistic Management* 30(10): 847-868

.LRN TM (May 22 2012) ".LRN Home: Learn, Research, Network" [On line] http://dotlrn.org/

Minuteman Software. (May 22 2012) "Computer Simulation" [On line] http://www.minutemansoftware.com/

Moodle (May 22 2012) "All we want to do is to give you powerful free tools to help you educate the world" [On line] www.moodle.com

Paulsen, M. F. (2003). *Experiences with Learning Management Systems in 113 European Institutions*. *Educational Technology & Society*, 6 (4), 134-148

Sakai Project (May 22, 2012), "Collaboration and learning - for educators by educators". [On line] http://www.sakaiproject.org/

Ståhl, Ingolf. "GPSS: 40 years of development" Proceedings of the 33nd Conference on Winter Simulation 9 Dec. 2001: 577-585.

Storch, Matthew Francis, and Jane WS Liu. "A framework for the simulation of complex real-time systems." (1997).

Wolfgang K. and K. Osterbye, 1998. BetaSIM a framework for discrete event modeling and simulation. *Simulation Practice and Theory* 6(6): 573-599

Wolverine Software Corp. (May 22 2012) "Welcome to Wolverine Software!" [On line] www.wolverinesoftware.com/

Zikic, A. M.; Radenkovic, L. J.: *New Approach to Teaching Discrete Event System Simulation*. International Journal of Engineering Education; VOL 12; NUMBER 6; 457-466; 1996