# Discrete Event Simulator for Communication Networks

Elena Uleia[1], *member IEEE*, Octavian Fratu[2], *member IEEE*, and Simona Halunga[3], *member IEEE*

***Abstract* — Discrete event simulation is widely used as a simulation method. The communication networks can become very complex and time consuming to simulate. The objective of the OSsim toolset is the development of a network simulation and analysis package which makes use of technologies from the area of distributed computing, client-server architectures and object oriented programming. One of the key issues of the design is high performance of the overall system, making possible the modelling and the simulation of complex networks.**

***Keywords* — editor, GUI, network, simulations, toolkit.**

## I. Introduction

The objective of this paper is to present the OSsim (Open Source Simulator) software package main features, designed to provide a comprehensive work environment for the network modeller [1]. It can be used in diverse applications areas of communications networks such as:

- to measure the performances of existing or future conditions networks under a wide range of conditions;
- to analyse and simulate queuing systems to design,
- to debug and fine-tune discrete-event system models.

The targeted audience is primarily among groups from academic environments.

Alternate commercial tools do exist already on the market. Most known tools are OPNET™ (this stands for Optimised Network Engineering Tools), QNAP™. They are sometimes expensive, and almost always require intensive training.

The present paper is financially supported by the Romanian Ministry of Education and Research, under the research program CEEX no. 80/10.10.2005 SIUM and under the research program CEEX no. 240/2006 NEM-RO.

[1]Elena Uleia is Ph.D. student, Computer Science Faculty, University POLITEHNICA of Bucharest, Romania (phone: +40-21-4024996; fax: +40-21-3169622; e-mail: elena.uleia@gmail.com).

[2]Octavian Fratu is with the Telecommunications Dept., Faculty of Electronics, Telecommunications and Information Technology, University POLITEHNICA of Bucharest, Romania (phone: +40-21-4024996; fax: +40-21-3169622; e-mail: ofratu@elcom.pub.ro).

[3]Simona Halunga is with the Telecommunications Dept., Faculty of Electronics, Telecommunications and Information Technology, University POLITEHNICA of Bucharest, Romania (phone: +40-21-4024996; fax: +40-21-3169622; e-mail: shalunga@elcom.pub.ro).

## II. The Toolkit Design

### A. Basic Principles

Among the objectives of the project was not only the development of a tool comparable in performance and case of use with its commercial counterparts, but also with the ability to run on cheap hardware (mid-range PCs).

One approach would be the use of a software system (e.g. PVM, Parallel Virtual Machine, which would permit a heterogeneous collection of networked computers to be viewed as a single parallel computer.

Object Oriented Programming is used as support for a better organization of code and easy future reutilization of the software components.

### B. Simulation Side Design

Generally speaking, there are two trends in the world of simulators [2]: One way is to use a custom programming language which reflects at semantic level the targeted domain. An example in this respect is QNAP, which has a syntax reassembling Pascal and basic data types such as queues and traffic sources. In this case, the performance issues are heavily dependent on the implementation. If the language is interpreted and not compiled, it is likely to have lousy performance in terms of simulation time for systems in which events happens very fast (e.g. ATM switches).

Another way is to build a simulation environment around a classic compiled language, consisting at least in a library of user-capable routines and, in commercial products, in a graphical interface which hides the nonessential programming details and provides to the user the ability to visualise the structure of his/her target application. An example of such a tool is OPNET, which builds around the C language a high performance simulation environment used in many mission-critical applications [3].

OSsim follows the second approach, embedding C++ code in its core and thus allowing the user to enjoy the power and expressiveness of today's most successful programming language. Its basic simulation engine is designed as a C++ class hierarchy packaged as a class library; the simulation engine makes use of advanced features of the C++ language, as polymorphism and multiple inheritance. However, the average user needs not

to know any of this: Only a basic knowledge of C++ (and of course the Simulation API) is required in order to be productive, as the graphical interface (which will be introduced later on the client side) is responsible for the generation of the C++ code which implements the structure of the target application. Besides the simulation engine (also known as the Simulation Kernel), the server side features the Base Models Library, an extensible set of building blocks available to all registered users. The Base Models Library is developed within the same framework as the user model, but it cannot be altered by ordinary users.

Important to note is that both user code and the automatically generated code by the client side inherits the OSsim Simulation Kernel. In the final linking step, a combination of base and user models can be used to produce the simulation executable.

### *C. Graphical User Interface (GUI) Design*

OSsim package implements a Graphical User Interface (GUI) which is up to some extend, independent of the underlying OS, and windowing technology.

The current implementation is available for UNIX / X Window System platforms. A feature of the GUI is that it can be run in a client-server architecture, remotely from the simulator side, on completely different hardware architecture. Of course, it can be also run on the same computer with the simulator side if this architecture is preferred. In practice, we used both PCs and workstations for running the GUI.

## III. Tool Features

The follows is a list of the key features of OSsim:

**domain specific:** OSsim is designed specifically for the development and analysis of communications networks.

**hierarchical models:** Network models can be hierarchically structured, allowing the re-use of previously developed models in different simulations

**graphical specification of models:** Specifications are entered graphically with specialized editors. These editors provide an efficient medium for design capture via a consistent set of modern user-interface methods: mouse-driven menus and icons.

**automatic generation of executable simulations:** OSsim provides an efficient event-driven Simulation Kernel, libraries of models and service functions (via the Simulation API). It takes the design specification and automatically generates an executable simulation.

**simulation debugger:** An interactive debugger may be used to monitor model behaviour in detail. This is event debugging, allowing the user to set up breakpoints ‘on time’, but not on source code.

**analysis tools:** Evaluation and trade-off analysis require a large volume of simulation results. A set of analysis tools is provided to interpret them in a graphical form, and visualize these.

**modelling with C++ language:** Models processes are described with a hybrid approach which allows users to embed C++ language code with a graphically specialized Extended Finite State Machine (EFSM). The specification of processes in C++ is facilitated by a library of support functions which provide the most important simulation services.

## IV. The Simulation Model

Here below, are given some remarks about the theoretical framework upon which the entire system is built, in order to allow the reader to better understand the functionality of the user interlace.

Besides running on a single host architecture, as most programs do, OSsim can optionally be run in a distributed client-server architecture, in which the client is the graphical user interface (GUI) and the compilation and simulation services are provided by a remote server.

The theory behind the OSsim design is the Extended Finite State Machine (EFSM) model.

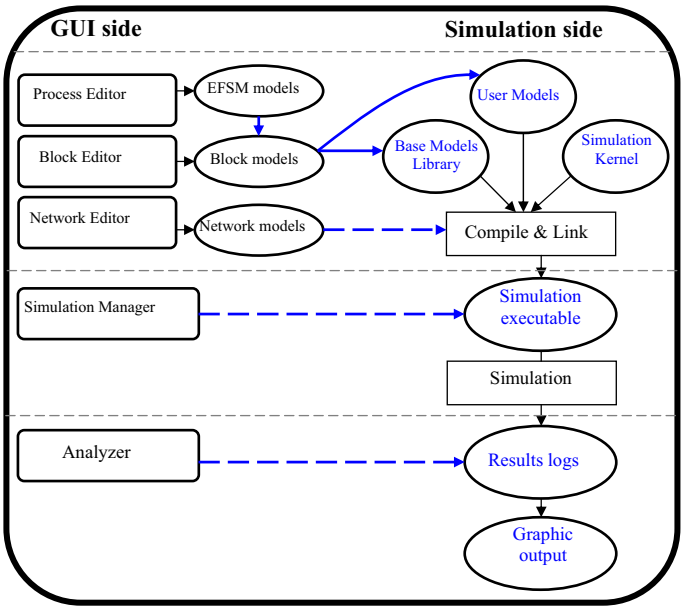


Fig. 1. OSsim internal interactions

The EFSM model can be thought as an entity which represents protocols, algorithms, or in general, decision making processes. We will not go in further details on EFSM as this is well covered in literature [4].

There are many simulation languages or environments which use the EFSM approach. EFSM instances are usually known as processes and can be thought as the "active" part of the modelled system. However, in order to express its structure, some hierarchical grouping rules and communication between the basic building blocks must be present. This is usually done at semantic level in the case of simulation languages and by graphical means for simulation environments.

What is also common to all EFSM based approaches is the parallel (simulation) of the processes, with regards to a fictive execution time axis called the simulation time. Fictive time means behaviour of the simulated systems studied, as opposed to the actual time needed for such a simulation to take place.

It is important to note that the simulation time is not a linear function of the real time, because the simulation is

event-driven and not incremental with a constant pace. That is, the periods of time for which nothing happens in the simulated system are skipped.

## V. The Toolkit Description

The toolset supported by the OSsim package covers three functional areas: Specification, Simulation, and Analysis.

The Specification area consists of a set of four graphical editors: the Process Editor, the Block Editor, the Network Editor, and the Parameters Editor. The Simulation area consists of the simulation user interface. The Analysis area is represented by a set of tools capable of statistical interpretation of the raw data obtained from simulation, and graphical representation.

OSsim features the Process Editor, a tool used for the graphical specification of process models. The specifications are based on EFSM representations, and include the names of states, transitions between states, the conditions for each transition, the actions which are taken upon entering or exiting a state or making a transition, temporary and state variables, and formal attributes of the process.

The basic building blocks and the more structurally complex ones are described in OSsim using the Block Editor. This tool is used to specify block models, which consists of a set (if any) of block objects (physical instances of already defined blocks) and optionally of an "active" part, which should be seen as an attachment and is represented by a process model instance. The basic "brick" is an empty block with a process attached. The Block Editor allows the user to interconnect in an arbitrarily complex graph the block objects, expressing the information flow between them.

The Network Editor is a modelled version of the Block Editor which also provides the operations necessary to bind together all lower level specifications into a single executable simulation.

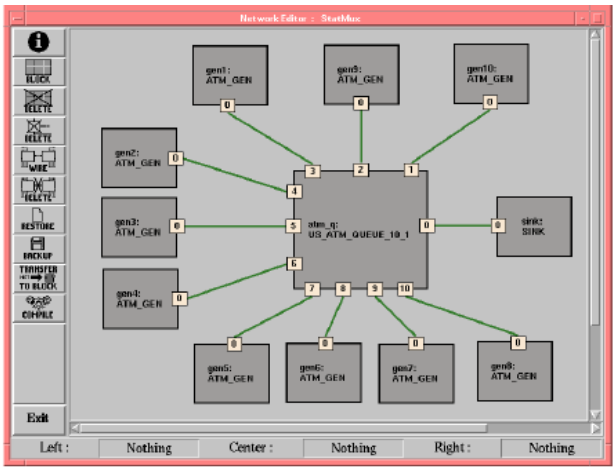


Fig. 2. Network Editor

The Simulation Manager provides an environment for setting up one or more simulation runs, specifying the input parameters, and directing their collected data into named output files. It can also initiate debug sessions, allowing the user to monitor model behaviour in detail.

The Simulator provides support for a series of statistical distributions, both discrete and continuous distributions. For continuous state variables, the implemented continuous probability distributions are: uniform distribution, and standard uniform distribution, exponential distribution, normal and normal square distributions. On the other hand, for any discrete state variables there are implemented the following discrete probability distributions: Bernoulli distribution, geometric distribution, Poisson distribution, constant distributions, and the convolution Erlang distribution [5],[6].

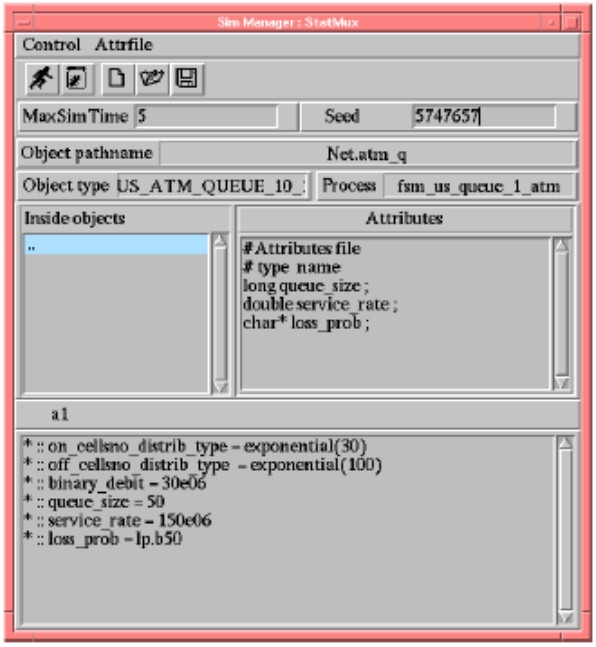


Fig. 3. Simulator Manager

Finally, the Analyser is used to analyse simulation result data, which can be plotted with a variety of graph types.

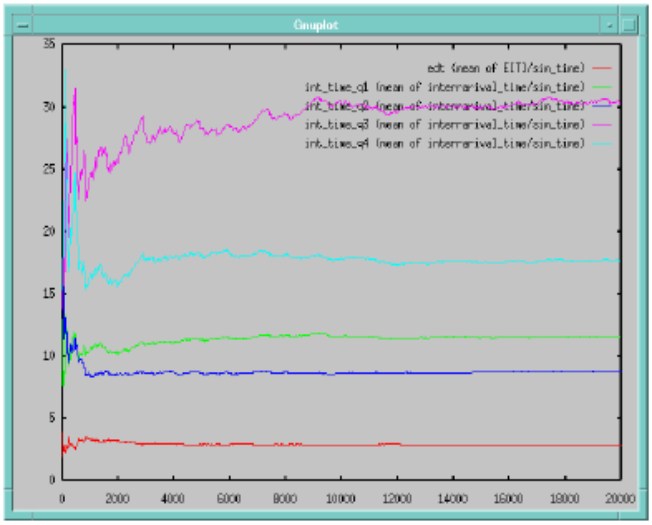


Fig. 4. The Graphical Analysis Results

## VI. Performance Report

The model used for evaluation is an ATM statistical multiplexer (ATM_Stat_Mux), which does very little I/O at the end of the simulation. This is particularly important if one wishes to evaluate the raw performance of the simulation kernel, because mixing significant I/O activity with computations degrades the overall performance.

Each event processed by the simulation kernel implies the invocation of an extended finite state machine.

The table I below shows the simulations time for an ATM statistical multiplexer.

TABLE I: SIMULATIONS TIME FOR ATM MULTIPLEXER

| Computer | CPU | RAM | Events/sec. | ET/1E10* |
|---|---|---|---|---|
| PC Desktop | Core Duo E6400 / 2.13GHz | 2Gb / cache L2 2Mb | 3.087E6 | 54min |
| Dell Laptop 6400 | Core Duo T7200 / 2.0GHz | 2Gb / cache L2 4Mb | 3.366E6 | 49.5min |

* estimated time necessary to simulate $10^{10}$ events.

It appears that the amount of available RAM has a limited impact on performance; what are really decisive are the processor, and the clock frequency.

## VII RESULTS AND CONCLUSIONS

Most importantly, the easy building of the target application allows focusing on problem and not on implementation details. The built-in support .for an easy to extend library of base models will surely boost productivity for the network modeller.

The high performance network simulation environment is capable of simulating, on usual hardware, 1.2E10 events within one hour, based on several implementation optimisations [7].

The current development release is l.0alpha6. The current body of code is relatively small – about 5,000 lines of C++ code for the Simulation Kernel, and about 8,000 lines of Tcl code for the GUI part.

However, the size of a program is by no means the right measure for its performance: In a benchmark involving a statistical ATM multiplexer, OSsim outperformed QNAP by a factor of 7 in terms of simulation time. The OSsim proved to be better in terms of execution speed by a factor of 15 for a simple queueing system comparing to OMNeT++ tool [8].

## REFERENCES


[1] Elena Uleia, 'OSSIM: A Switched Packet Network Simulator', U.P.B. Scientific Bulletin, to be published.

[2]. D.C.Schmidt, 'Model-Driven Engineering', in IEEE Computer Magazine, Feb.2006, pp.25-31

[3]. W.Kreutzer, 'System Simulation: Programming Styles and Languages', Addison-Wesley, 1986

[4] G.J.Holzmann, 'Design and Validation of Computer Protocols', Prentice Hall, 1991

[5]. J. Banks, J.S.Carson II, B.L. Nelson, 'Discrete-Event System Simulation', Prentice Hall, 1999

[6]. U.W.Pooch, J.A.Wall, 'Discrete-Event Simulation: A Practical Approach', CRC Press, 2000

[7]. Elena Uleia, Simona Halunga, O.Fratu, 'Techniques for Implementing Real-Time Protocols for Mobile Communications Systems', IEEE TELSIKS 2005 Conference, September 27-30, 2005, Nis, Serbia and Muntenegro, published in Proc. of TELSIKS 2005, pp. 85-88, ISBN 85-85195-28-4.

[8]. Elena Uleia, 'Performance Evaluation for Discrete Event Simulators: OSsim and OMNeT++', U.P.B. Scientific Bulletin, to be published.