OSSIM: A SWITCHED PACKETS NETWORK SIMULATOR

Elena ULEIA[1]

Pachetul de programe OSSim reprezintã un mediu software de simulare şi analizã a reţelelor, în special, şi a oricãror procese stochastice din sistemele discrete, în general. Acesta înglobează technologii din domeniul calculului distribuit, arhitecturi client-server, cât şi a programării orientate obiect. Unul din punctele cheie ale proiectării sistemului de faţă, este pefrormanţa globală înaltă a acestuia, ceea ce face posibile modelarea şi simularea unor reţele complexe.Utilitarul OSSim este o implementare complet nouã, bazat pe sursã de cod UNIX 'open-souce' , fãrã a refolosi sursã de cod al unui alt simulator existent.

The OSSim software package is a development environment created for supporting network simulation and analysis, in particular, and any stochastic processes of discrete systems, in general. It 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.The OSSim toolset is a brand new implementation, based on UNIX open source, not making use of any existent source code from any other simulators.

Keywords: simulation, protocol modelling, queuing systems, performance analysis, hierarchical models, distributed computing

1. Introduction

The objective of this paper is to present the OSSim (Open Source Simulator) software package, designed to provide a comprehensive work environment for the network modeller. It can be used in diverse application 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.

The OSSim toolset has been entirely designed and implemented by the author within the PhD thesis scope at the Computer Science Faculty, spread over several years of study. The PhD thesis dissertation is covering methods for

[1] PhD student, Faculty of Computer Science, University POLITEHNICA of Bucharest, Romania

validation and simulation of distributed real-time systems. This tool has been designed as a discrete events-oriented real-time distributed system in itself.

OSSim has been intended to be an engineering tool capable of simulating large communications networks with detailed protocol modelling and performance analysis; However, any system with a hierarchical structure which admits a discrete time modelling also fits in.

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

2. OSSim key 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 is implemented for monitoring the model behaviour. 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.

3. The toolkit design

3.1 OSSim simulation description

Generally speaking, there are two trends in the world of simulators [1]: 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 [2].

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 do inherit the OSSim Simulation Kernel class. In the final linking step, a combination of base and user models can be used to produce the simulation executable.

3.2 Graphical user interface

OSSim package implements a Graphical User Interface (GUI) which up to some extent is 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.

4. OSSim simulation model framework

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 interface.

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.

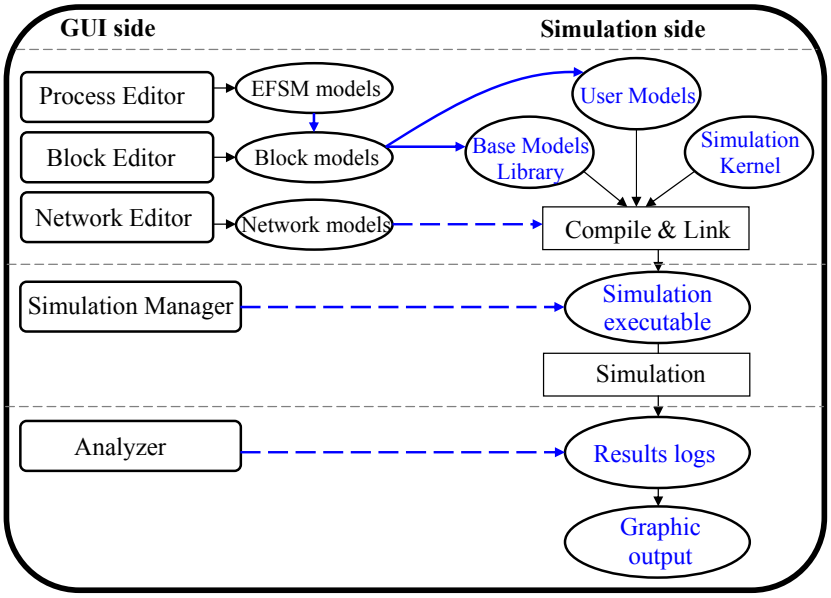

Fig. 1. OSSim internal interactions

The theory behind the OSSim design is the Extended Finite State Machine (EFSM) model. 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 [3].

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.

5. OSSim toolkit implementation

In order to build an executable simulation program, one first needs to translate the GUI user interface modules files into C++, using the internal parser. After this step, the process is the same as building any C/C++ program from source: all C++ sources need to be compiled into object files (.o files on Unix/Linux), and all object files need to be linked with the necessary libraries to get an executable. The attributes file is loaded dynamically at execution time.

The models processes are described with a hybrid approach which allows users to embed C++ language code with a graphically specialized EFSM. The specification of processes in C++ is facilitated by a library of support functions which provide the most important simulation services.

A simulation system is made up of blocks, which can have ‘infinite’ level of hierarchy, and processes on top of the simulation kernel. The blocks encapsulation starts from bottom up, based on a BaseBlock class.

The blocks describe the structure of the system, and the processes the execution engines. Not all blocks have attached processes. Once a process is attached to a block, the user block will have multiple inheritance, inheriting from the process class too.

The BaseBlock class creates actually the startup framework. By adding blocks at the same level of encapsulation, or at higher level, one defines the simulation framework structure, creating a block template for the entire system. The blocks are instantiated using a mechanism of ‘templates’ based on pre-processor. The pre-processor directives are used to allow re-use of block classes

and to create specific block instances. They are also used for inclusion of automatically generated code from the GUI interface.

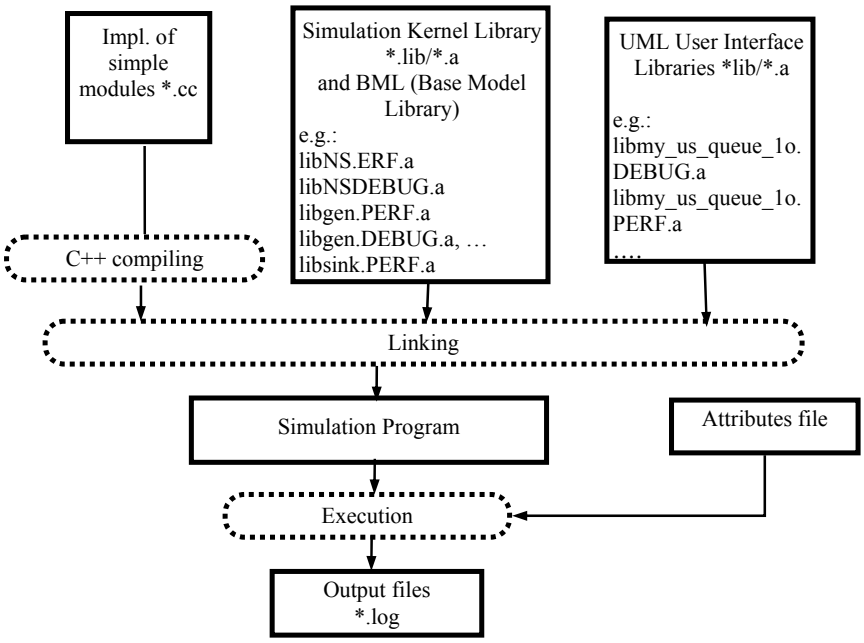

Fig. 2. Building and running an OSSim simulation

The applied encapsulation rule is to create a class based on a base class, with members of block types that instantiate other object types. These classes inclusion is such as structures variables. Classes are this way automatically generated based on the template class.

6. OSSim experimental results

Methodologies for establishing model credibility, statistical analysis and experimental designs are currently largely unsupported by simulation software. More guidance could be given, particularly in interactive environments. A small body of well-established techniques is available, although mostly scattered through journals and conference proceedings. It is to be hoped that some of the more robust of these will be integrated with new simulation systems, possibly augmented by some 'intelligent' assistance [4].

In [5], Lang has designed and developed a library for resource based conceptual models. This approach is implemented by most non object-oriented simulation frameworks such as Arena, and a few object-oriented frameworks such

as SimKit and DSOL. In this approach one designs the model as a chain of stations.

OSSim is using the simulation kernel as the main class in the hierarchy, whereby the simulation model classes, for both blocks and processes (EFSMs) do inherit from.

6.1 Queueing network model example

A queueing network model has been designed using the OSSim toolkit by means of a top level system block.

The main components of the system model are the:

- three uni-servers (q12, q22, q32. First index is the inputs line number, and the second index is the outputs line number, respectively)
- one traffic generator (gen), with two output lines (‘0’, ‘1’),
- one real channel with delays (‘*real_ch*’), and
- two sink blocks (sink1, sink2), with one entry gate each (‘0’).

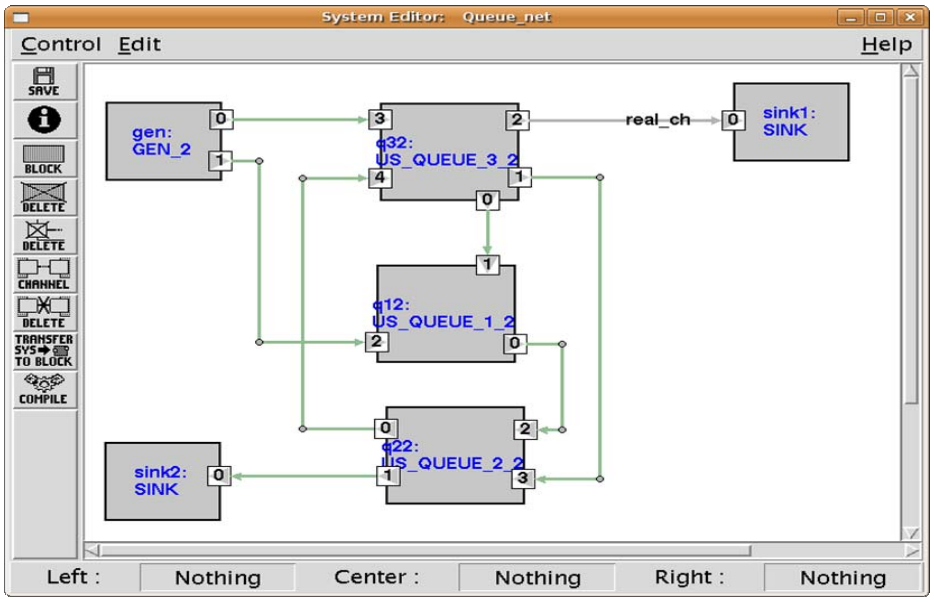

Fig. 3. Queueing model using OSSim

Network models can be hierarchically structured. Each model entity is represented by a block with specific input and output gates. Every block is an object instance of a specific block type. E.g. the uni-server q32 is an object instance of US_QUEUE_3_2 block type, which defines a two-entry and three-exit uni-server type.

Each entity block is an EFSM, denoted by an attached process. A real channel ('real_ch') does exist between gate_2 of q22 and gate_0 of sink1. This has an attached process that manages, in its EFSM, the channel properties.

6.2 Queueing network model simulation results

In order to prove the stability of the whole system, the following convergence equation should hold for each queue in the system: inter-arrival mean time value at queue *i*, multiplied by the ratio of the number of packets received at current queue *i* ports and total number of the generated packets should equal the mean value of the generated packets in the system.

$$\lambda = \lambda i \frac{L_i}{L} \tag{1}$$

where, λ – mean arrival time of the generated packets;

λ_i – inter-arrival mean time at queue *i* ;

L – total number of generated packets injected in the system during simulated time T;

L_i – number of packets arriving at queue *i* input ports, during simulated time T.

The queues inter-arrival mean times parameters are proved both graphically and analytically by means of the equation (1) calculations, given in table 1 below.

Tabel 1

Mean inter-arrival times - computed and experimental values

Convergence results	**No of samples of queue *i* [N_i]**	**Mean of interarrival time of queue *i* [λ_i]- *computed***	**Mean of interarrival time of queue *i* [λ_i]- *experimental***
us_queue_q12	833155	1.200590	1.200241
us_queue_q22	1066881	0.937572	0.937311
us_queue_q32	931978	1.0732849	1.0729815
us_queue_obs1	1000278	1	0.9997155
us_queue_obs2	1000278	1	0.9997155

From the above table, the q12 received packets are 833155+1, the total generated packets number is 1000278+1, and the generated packets distribution mean value is 1. By applying equation (1) we get the queues inter-arrival mean time as:

$$\lambda = \lambda i \frac{L_i}{L} \quad => \quad \lambda i = \lambda \frac{L}{L_i} \tag{2}$$

$$\lambda 1 = \lambda \frac{L}{L_1} = 1 * 1000279 / 833156 = 1.200590$$

$$\lambda 2 = \lambda \frac{L}{L_2} = 1 * 1000279 / 1066882 = 0.937572$$

$$\lambda 3 = \lambda \frac{L}{L_3} = 1 * 1000279 / 931979 = 1.073284$$

The experimental results are very close to the computed results, which prove the convergence of the system. For the current simulation time, that is 10^6, the convergence deviation is about 2.8 10^{-4} for all computed inter-arrival mean times. A null convergence deviation is to be obtained for very long simulation time. For example, a simulation run of 10^8-10^9 is considered sufficient for the current queueing system.

7. 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.

Tabel 2

Simulations time for an ATM statistical 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.

8. 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 simulation executable is practically statically generated with little dynamic information (the parameters in the configuration files, seed, and simulation time), which makes it very fast. The use of the model template class eliminates any dynamic lookups in a hash table for the EFSM states as it happens in most C++ simulations, or other application tools.

An interactive simulation debugger has been developed to monitor model behaviour in detail. The simulation debugger is actually a process that has traces attached to it. This is automatically attached to the simulation system, when the simulation executable is generated.

Another idea applied in this implementation, was to allow actions to occur on transitions, not only in the process states. This is a design decision to improve the model internal behaviour.

The high performance network simulation environment is capable of simulating, on usual hardware, 1.2E10 events within one hour, based on several implementation optimisations [6].

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 FIFO queue system, OSSim outperformed OMNeT++ by a factor of 15 in terms of simulation time.

R E F E R E N C E S

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

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

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

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

[5]. *Lang, Neils A.* & all, 'Distributed open simulation model development with DSOL services'. In: ESS'2003, Proceedings 15th European Simulation Symposium 2003 – Simulation in Industry, (Delft, The Netherlands, October 2003), SCS European Publishing House, Germany, (ISBN 3-936150-28-1), pp. 210-218., 2003.

[6]. *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.