EVENT BEHAVIOR OF DISCRETE EVENT SIMULATIONS IN CD++ Vs. NS-2

Shafagh Jafer, Gabriel Wainer
Dept. of Systems and Computer Engineering
Carleton University Centre of Visualization and Simulation (V-Sim)
1125 Colonel By Dr. Ottawa, ON, Canada
{sjafer,gwainer}@sce.carleton.ca

Juan-Carlos Maureira Bravo, Olivier Dalle
Université de Nice Sophia Antipolis, CNRS, and INRIA
06902 Sophia Antipolis Cedex, France
{Juan-Carlos.Maureira_Bravo, Olivier.Dalle}@sophia.inria.fr

Keywords: DEVS, CD++, NS-2, FES, ELT.

Abstract

The study of events behavior through real simulations could contribute to develop or improve Future Event Set (FES) data structures in order to achieve better performance on large scale simulations. In this paper we have analyzed FES data structures of two discrete event simulators: CD++ and NS-2. We have run variety of simulations on each simulator to describe a real event behavior by observing event timestamps, life times into the FES and firing time (event execution time). The goal of this research is to present new ideas on how the FES data structures could be improved exploiting event behaviors.

1. INTRODUCTION

The Future Event Set (FES) is an important component of discrete-event simulations, as it handles the future events to be executed (or fired) by the simulator core. A poor implementation of this data structure could easily degrade the overall performance of the simulator. Given that, to contribute to the current evaluation framework, this paper will attempt to explore event behavior by a CD++ simulator instrumentation in order to present some empirical results and, eventually, discuss about how events behavior can be exploited in order to improve a FES data structure. On the other hand, the results are compared to those obtained from running similar experiments on NS-2.

The main part of this research was running simulations on CD++ (DEVS and Cell-DEVS hierarchical simulator). This was performed by running different DEVS models under different circumstances and collecting the data which were mainly associated with FES characteristics. Analyzing DEVS simulator, CD++, is more complicated compared to NS-2 because there is no centralized FES in CD++, so a virtual FES has to be computed a posteriori. Another main difference among CD++ and NS-2 is that the first one uses "simulation time" as opposed to latter one which uses "real-time". This implies that, when for instance, when an event has to stay in FES for two seconds, it does actually stay there for exactly two seconds in NS-2, while in CD++ it only stays in FES for a much shorter time, depending how busy the simulator is. More details about the structure of CD++ will be provided in the next section.

2. BACKGROUND

The evaluation framework that was used to evaluate the FES of NS-2 is the Hold model [1]. According to it, two distributions must be chosen: the scheduling distribution (1) and the future event set distribution (2). The first one is related to the time where new events are enqueued into the FES, and the second one is related to how long an event will remain into the FES (living time). With these distributions, several hold operations are performed: a new event is created according (1), and it is scheduled to be fired according (2). The process is repeated n times (n Hold operations). When one of the already created and scheduled event is fired, a new event is created and scheduled according (1) and (2), completing the cycle. The distributions mostly used to build a hold operation are the uniform, triangular, reverse triangular and the camel (combination of beta functions [2]).

DEVS [3,4] is a formalism for modeling and simulation of DEDS (Discrete Events Dynamic Systems) which provides a framework for the definition of hierarchical models in a modular way by decomposing the real system into behavioral (atomic) and structural (coupled) components. DEVS theory provides a rigorous methodology for representing models, and it does present an abstract way of thinking about the world with independence of the simulation mechanisms and the underlying hardware and middleware. A DEVS atomic model is formally defined by:

$M = \langle X, Y, S, \delta int, \delta ext, \lambda, ta \rangle$,

where

$X = \{(p,v) \mid p \in IPorts, v \in Xp\}$ is the set of input ports and values;

$Y = \{(p,v) \mid p \in OPorts, v \in Yp\}$ is the set of output ports and values;

S is the set of sequential states;

$\delta int: S \rightarrow S$ is the internal state transition function;

$\delta ext: Q \times X \rightarrow S$ is the external state transition function, where

$Q = \{(s,e) \mid s \in S, 0 < \in < ta(s)\}$ is the total state set, e is the time elapsed since the last state transition;

$\lambda: S \rightarrow Y$ is the output function;

$ta: S \rightarrow R^{+}_{0,\infty}$ is the time advance function.

The semantics for this definition is given as follows. At any time, a DEVS coupled model is in a state s S. In the absence of external events, the model will stay in this state for the duration specified by ta(s). When the elapsed time e, is equal to ta(s), the state duration expires and the atomic model will send the output λ (s) and performs an internal transition to a new state specified by δ_{int} (s). Transitions that occur due to the expiration of ta(s) are called internal transitions. However, state transition can also happen due to arrival of an external event which will place the model into a new state specified by $\delta_{ext}(s,e,x)$; where s is the current state, e is the elapsed time, and x is the input value. The time advance function ta(s) can take any real value from 0 to ∞. A state with ta(s) value of zero is called transient state, and on the other hand, if ta(s) is equal to ∞ the state is said to be passive, in which the system will remain in this state until receiving an external event.

Cell-DEVS extends DEVS formalism, allowing the implementation of cellular models with timing delays. Two types of timing delays can be used, namely transport and inertial [5]. When transport delay is used, the future value is added to queue sorted by output time, allowing the previous values that were scheduled for output but have not yet been sent to be kept. On the other hand, inertial delays allow a preemptive policy at which any previous scheduled output value will be deleted and the new value will be scheduled. A Cell-DEVS atomic model is defined by [6]:

$$TDC = < X, Y, I, S, \theta, N, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D >$$

CD++ [7] is a modeling tool that implements the DEVS and Cell-DEVS theories by applying the original formalisms. The toolkit includes facilities to build DEVS and Cell-DEVS models. DEVS atomic models can be programmed and incorporated into a class hierarchy programmed in C++. Furthermore, coupled models can be defined using a built-in specification language. Therefore, coupled and Cell-DEVS models need not to be programmed, rather the tool provides a specification language that defines the model's coupling, the initial values, the external events, and the local transition rules for Cell-DEVS models.

3. DEVS MODELS

The experiments on CD++ were carried out by running various DEVS models. In this paper two of the models namely, Alternating Bit Protocol (ABP) [8], and Discrete Event (DE) Controller [8] will be presented in details. The following is a brief description for each model.

3.1. ABP Model

ABP (Alternating Bit Protocol) is a communication protocol to ensure reliable transmission through unreliable network. The sender sends a packet and waits for an acknowledgement. If the acknowledgement doesn't arrive within a predefined time, the sender re-sends this packet until it receives an expected acknowledgement and then sends the next packet. In order to distinguish two consecutive packets, the sender adds an additional bit on each packet (called alternating bit because the sender uses 0 and 1 alternatively). A DEVS model called "ABP Simulator" is created to simulate the behavior of the Alternating Bit Protocol.

The ABP Simulator consists of 3 components: sender, network and receiver. The network is decomposed further to two subnets corresponding to the sending and receiving channel respectively.

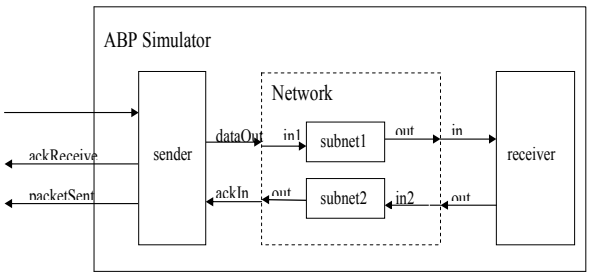

Figure 1. Structure of ABP

3.2. DEController Model

The DEController represents Discrete Event Multiple Model Control of a Time Varying Plant. Conventional adaptive control using a single identification model, is efficient when the initial parameter estimation error is small, and plant parameters are slowly varying over time. The use of multiple models becomes appropriate, when either of these conditions are not satisfied, such as in the case of a subsystem failure or a change in the operating environment.

Typically, a finite number of models are evaluated by an index-of-performance, where, at any instant, the most suitable model's parameterized controller is applied to the plant. This approach proves beneficial for maintaining control of a plant when there is parameter jumps. Additional discussion of multiple model control for continuous or discrete time systems is provided in Reference [9].

Multiple model control demands a union of high-level decision making with mathematically complex algorithms. Implementing such, using discrete-event math is where much of this paper's design is focused. Once the theory and testing was performed, was an implementation in CD++ modeling language attempted.

4. EXPERIMENTS

Every DEVS model created in CD++ consists of a C++ header file (name_of_atomic_model.h) and a source file (name_of_atomic_model.cpp) per atomic model. Aside, for every coupled model including the TOP most model, an MA file is required. CD++ provides a virtually real-time –based simulation environment which although is not real-time, but it appears to be. This is the main difference between CD++ and real-time simulators such as NS-2. In NS-2 the

messaging and waiting times are based on real-time. For instance, in CD++ when simulating a model, the run time can be specified prior to the simulation, so if for example we specify the simulation run time to be 1 hour, it turns out to run for a couple of seconds and not 1 hour. However, the messages will be presented in a manner that the simulation actually took 1 hour. In contrast, NS-2 simulations run exactly for a specified real-time. Therefore, in NS-2 the user has to wait exactly for that amount of time in order for the simulation to end and see the results. While in CD++ the user waits for a virtual time although the simulation virtually takes "real-time" long.

4.1. CD++ Simulations

Two sets of experiments were collected for each DEVS model:

1. Using predefined holdIn duration,
2. Using random holdIn duration.

The predefined holdIn duration is the normal behavior of each DEVS model, meaning that, before simulation begins, each atomic model defines the amount of time it will spend acting on an input (time spent in holdIn) as a predefined time.

The reason behind running simulations for both predefined and random holdIn times is to avoid getting same set of timing behavior. This is due to having predefined processing time of atomic models on different inputs, which in turn results in having same set of messages to get repeated in a cyclic pattern which is not useful information. Example of these patterns will be reflected in the following section when the simulation results are discussed.

As mentioned at the beginning of the paper, the results of this research are analysed with respect to those achieved from NS-2 simulator. Since NS-2 uses real time simulations, we have implemented this feature in part of CD++ to find out the exact wait time in holdIn (our virtual FES) in real-time measures. This was performed by creating a Timer class that uses system's clock ticks and frequency to compute the real time duration of holdIn function execution.

By implementing this into our atomic models source code, we get real-time –based durations spent by each atomic model's done messages (the time the atomic model spent in holdIn function). As we pointed out in previous sections, out holdIn function serves as a virtual Future Event Set which defines the duration for which an atomic model was acting on inputs or was passive and waiting for new inputs.

The real-time is computed as milliseconds and it is inserted right before a call is made to holdIn() and right after return from this call. Represents adding this computation to our existing atomic model's source code.

4.2. NS-2 Simulations

For Ns-2 the simulations were carried on based on the following methodology:

In order to discover the distribution of event execution time and life time into the FES, some tracing code were introduced on the NS2 core simulation (scheduler mostly) in order to collect information about the event on the FES. A Calendar Queue was used as scheduler, in order to evaluate big simulations in a reasonable time. The FES event length were recorded in fixed intervals, the enqueues and dequeues were counted, and also all the event living time into the FES (overall process and by snapshots) were traced. With the obtained data, using statistical methods, the empirical distribution of the FES was built. With that, in addition with the enqueue/dequeue patterns (scheduling distribution), some interesting information about how the event behaves along the simulation time flow can be concluded.

The Test bed to collect information about event behavior was around 100 simulations from different sources, as published papers, NS-2 examples and performance tests.

5. RESULTS

In this section the results of simulations on both simulators, CD++ and NS-2 are presented. The NS-2 simulations were carried by a research group from INRIA, France.

5.1. NS-2 Simulation Results

For each simulation, graphics were prepared to show the empirical distribution of the Event Fire Time (EFT) and the Event Life Time in the FES (ELT), considering all the events on a single run [10]. Also, snapshots of the FES were taken on regular intervals in order to explore, also, the evolution of the EFT and the ELT on the time. Here are the most representative graphics of each simulation (and replicas). The following graphs represent sample simulation results collected from NS-2 simulator under the described test-bed.

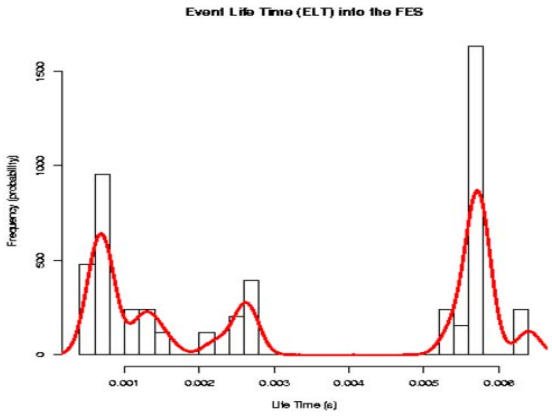

Figure 2. NS-2 sample simulation result.

5.2. CD++ Simulation Results

The simulations were carried out for each DEVS model under two different scenarios:

1. Random and non real-time holdIn durations,
2. Random and real-time holdIn durations.

Then, all the results were combined under real-time, and simulated time (non real-time) categories and the following statistical results were extracted.

Avg. simulTime (s)	Avg. realTime (ms)
8.3656	0.00189

The above numbers represent the average simulation time versus the average real time of all events life time that were spent in holdIn (FES) among all DEVS models. The total number of calls made to holdIn() was 1265. For each case, the data were plotted on a histogram to better analyze the distribution. Figure 3 and Figure 4. Correspond to these scenarios.

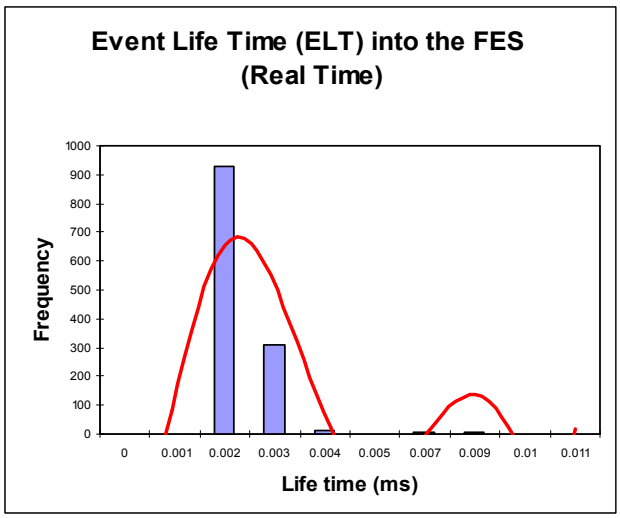

Figure 3. Real-time HoldIn durations

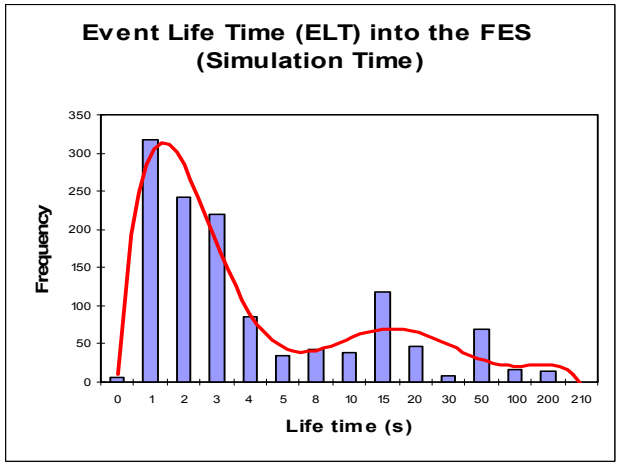

Figure 4. Simulated-time HoldIn durations

6. CONCLUSION

In this study, new interesting facts appears, as the event clusterization around the same wait times in the FES. This fact can be used to create new FES data structure that exploits this behavior. However, more simulations and experiments are needed to better judge the data. Also, the results obtained form CD++ and NS-2 simulators studies can be used to study the effect of hierarchical or flat architecture on the simulator's event handling behavior.

7. REFERENCES

[1] Chou C., Bruell, S., Jones D. " A Generalized Hold Model". In Proceedings of the 1993 SCS Winter Simulation Conference. Pages 756-761. 1993.

[2] A Jeff S. Steinman. "Discrete-event simulation and the event horizon". Proceedings of the eighth workshop on Parallel and distributed simulation. Pages 39-49. 1994.

[3] Zeigler, B. "Theory of modeling and simulation". First Edition. Wiley. 1976.

[4] Zeigler, B.; Kim, T.; Praehofer, H. "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems". Academic Press. 2000.

[5] Giambiasi, N.; Miara, A. "SILOG: A practical tool for digital logia circuit simulation". Proceedings of the 16thD.A.C. San Diego. 1976.

[6] Wainer, G.; Giambiasi, N. "Timed Cell-DEVS: modeling and simulation of cell spaces". Invited paper for the book Discrete Event Modeling & Simulation: Enabling Future Technologies. Springer-Verlag. 2001.

[7] Wainer, G. "CD++: a toolkit to develop DEVS models". Software - Practice and Experience. Vol. 32, pp. 1261-1306. 2002.

[8] Cell Based Discrete Event Simulation website. Available at: http://www.sce.carleton.ca/faculty/wainer/wbgraf/. [Accessed February, 2007].

[9] K. S. Narendra, O. A. Driollet, M. Feiler, and K. George, "Adaptive control using multiple models, switching and tuning," *Int. J. Adapt. Control Signal Process.*, vol. 17, 2003, pp. 87–102.

[10] Osa Wiki. Available at: http://osa.inria.fr/wiki/OsaPriv/ED-Diffserv. [Accessed February, 2007].