Luleå University of Technology
Computer Science
Håkan Jonsson

February 14, 2012

"SimBiz^{TM}! - Simulate your own Business!"

– Lab 5 in D0010E –

1 Background

One morning the company news letter contains an article that catches your eye. It seems that much to the surprise of top managers at *Acme Programming, Inc.*, the brand new product "Breadth First Graph Traversal" turned out to sell substantially less then expected (nothing at all, the rumour goes). The main cause was – they reason – purely bad timing. It was released the very same week a new expansion pack of the world's best-selling computer game of all times hit the streets. The latter is a "simulator" devoted to relations and "home decoration".

– *Reality is out, virtual simulation is definitely in!* – a slick and post-modern top executive says with a grin while staring back at you from a photo.

The article goes on to describe that in a brave attempt to save the company (and their own necks) the top management have decided to develop a very general simulator that will once again put the company back on track. You get mixed feelings when reading all this.

It was almost like you knew it was going to hit you. After lunch the very same day you find a memo in your pigeon-hole instructing your team (that's you and your fellow members that wrote encryption commands for *SimpleShell*) to embark on a new adventure. You get together and read the rather short instruction.

It seems that your task is to write a program that can be used to perform discrete event-driven simulations. Using this program, and some additional code you have to write, you should then simulate a car wash. A large part of the instruction is about the second half but it starts with a general description of the simulator itself.

2 A General Discrete Event-Driven Simulator

A discrete event-driven simulator is a program that makes it possible to study how a state S changes over time as events occur. It starts with an initial state and a start event, where the start event is the first member of a set of future events the simulator maintains during a run. It is appropriate to represent the set of future events with a *priority queue* where the priority is the time the events will occur. This is because we need to access future events in the order that they occur. The state should always contain a stop flag that signals the end of the simulation. Note that since this is a simulation, time is virtual and will be computed using formulas based on assumptions and involving randomness.

Events are implemented as Java classes that have a method (called, for example, *execute()*) that can be called to make them occur. This method typically changes the state S and generates new, future, events that are inserted into the set of future events. When a new event is created, the time in the future when it is to occur has to be computed. This is done in the event.

The simulator class contains a main loop that repeatedly extracts events from the set of events and "executes" them (makes them occur). The simulation stops when the set is empty or the stop flag in the state is set.

Note that the simulator is *independent* of the particular simulation it performs and that this has implications on the design. No part of the design of the simulator itself must depend on details of a particular simulation (a simulation of a car wash, for instance). The design should also take into account that it should be possible to have two or more simulations running (with separate states) at the same time.

3 Simulating a Car Wash

Based on the general simulator program you should then add specific pieces so that it is possible to simulate what happens at a car wash under various circumstances.

The simulated system (the car wash, that is) has a queueing system, k_1 fast car washing machines and k_2 slow car washing machines. When a customer with a car arrives, and a car washing machine is free, the car is immediately served. Otherwise, the customer has to wait in line with the car in the driveway that has room for at most k_3 cars. When a car washing machine becomes free, the next customer in line is served. If both fast and slow car washing machines are available, the fast machine is chosen. If the driveway is full of waiting cars, arriving customers will not be able to wait in line, and will therefore go looking for another car wash.

3.1 Time

To be able to simulate this system we need to model when new customers arrive and how long it takes to wash their cars. We assume that all fast machines are equivalent, as are all the slow machines.

In real life the distribution of arrivals depends on time of day, day of the week, and many other things. In our simulation we will assume that the expected value for the number of arrivals during a time interval is directly proportional to the length of the intervals, and that any two arrivals are two independent events. Statistics tells us that the time between two consecutive arrivals is exponentially distributed with the expected value y where $1/y$ is the expected value for the number of arrivals per time unit. The time it takes to wash a car could be expected to be normally distributed, but in this experiment we assume that it is uniformly distributed on the interval $[a, b]$. The only reason for this is that we want to avoid negative times.

It could be interesting to use a simulation like this to calculate the mean queueing time for the cars, how many cars that are denied service when the wait line is full, and the total amount of time that the car washing machines are idle. To design and interpret such simulations professionally, knowledge in statistics and queueing theory is required.

3.2 Random number generators

The simulation requires some sort of random numbers for arrival times and serving times. It is common to use a simple arithmetic formula to generate a sequence of "random" numbers. Since a formula is being used, the numbers are not random at all except possibly the first number in the sequence that can be chosen depending on the current time. The formula that is normally used is

$$r_i + 1 = (a \cdot r_i + b) \textbf{ mod } m$$

where a, b, and m are numbers chosen so that the sequence $r_0, r_1, \ldots$ goes through all integers in the interval $[0, m-1]$ in a way that appears random. That they actually aren't random can be seen by the fact that every other number is odd and every other number is even. It is therefore not possible to use the last binary number in each number to get a random stream of ones and zeroes. The number r_0 is called a seed of the sequence. A class representing a random number generator should have two constructors: one that takes an argument as a seed and one that uses the time of the method call as a seed. With a given seed you get a simulation that can be reproduced.

The class java.util.Random is based on such a random number generator with $a = 25214903917$, $b = 11$ and $m = 248$, where the last 16 bits are not returned. Random number generators for exponential and uniform distributions can be found on the course web page. Use these.

3.3 Events

The simulator should be driven by events. The events that can occur are that a car arrives to the system and that a car leaves the system when it has been washed. At these events the *state* of the system should change. The state is described by a time, a number of available car-wash machines, and a queue of customers that waits to be served. An arriving customer should be placed in the queue if all the machines are busy (if the queue is too long, the customer is rejected). Otherwise he should be served immediately, which means that the number of available machines of that type (fast or slow) should decrease by 1. Using the random number generator that produces serving

times, it is possible to calculate when the customer will leave the system. This future event should be added to the set of events maintained by the simulator.

When a customer leaves the system, there are two options. If the waiting line is empty, the number of available machines should be increased by 1. Otherwise the car that is first in line should be served, and the event representing that car leaving the system should be added to the set of events.

While it is possible to generate all event times times before the simulation is started, it is more rational (and required in this lab) to generate future events as late as possible during the simulation. This means that the program needs less memory without being less efficient. Implement your program in this more efficient way.

To stop the simulation, use a stop event. Once the stop event occurs, the simulation should stop immediately even if there are still events waiting in the queue.

The number of available washing machines and the current time should be represented by primitive variables in the state. Remember to distinguish between the fast and the slow machines! The only times that are interesting are those when an event occurs. This information should be contained in the priority queue. It can therefore be appropriate to keep track of the time in connection with the priority queue, and have the current time be the same as the time of the last event.

To be able to identify individual cars in the program they should be numbered sequentially from 0. The number assigned to a car should be unique. The line of waiting cars in the driveway should of course be represented with a Queue class.

4 Output

During the simulation of the car wash, the program should report all events. The simulation parameters should be printed first (for example, when the start event is processed). At the end of the simulation (when the stop event is processed), the total idle machine time, the total and mean queuing times, and the number of rejected cars should be printed.

For other events (when a car arrives or leaves), the following information should be included in the print-out:

- the time of the event
- time counters: the total idle time (for all washes combined) and the total queue time (if two cars are in the queue at the same time, this time is counted twice, etc.);
- the number of fast and slow car washes available BEFORE the event is processed;
- the number of cars in the queue and the number of rejected cars BEFORE the event is processed;
- the type of the event and the car ID.

You can use the class java.util.Formatter to format the time values. Figures 1 and 2 are a examples of output from a simulation using the following parameters:

- For car arrivals $ExponentialRandomStream(2, 1234)$ and $ExponentialRandomStream(1, 5, 1234)$,
- for fast machines $UniformRandomStream(2.8, 4.6, 1234)$ and $UniformRandomStream(2.8, 5.6, 1234)$, and
- for slow machines $UniformRandomStream(3.5, 6.7, 1234)$ and $UniformRandomStream(4.5, 6.7, 1234)$, respectively.

The number of car washing machines were in the first example 2 of each kind, in the second 2 fast and 4 slow, and the stop time was after 15 time units of simulation time. The queue had in the first example room for at most 5 cars at a time, in the second the maximum was 7.

5 Design

Before getting started on the simulation software, you have been ordered to first produce a design of the program. You think it might be wise to get some direction from the two lead designers of *Acme Programming, Inc.* They turn out to be notoriously hard to find, but one day you manage to corner them in the company cafeteria. They do not seem too happy to be disturbed while eating, but nevertheless give you some advice on the program, which should consist of:

1. A general Discrete Event-Driven Simulator (DEDS) that contains
   - Something that keeps track of events and always can say which event is about to happen.
   - Something that repeatedly makes the next event that is about to happen happen.
   - The *model*, representing the state that is changed by the events.
   - The *view*, responsible for presenting the state in some way.
   - Something that describes an "event" in an abstract way, so that it's possible to talk about an event without actually knowing what effect it has.
2. Application specific parts in terms of
   - A specific car wash state, which contains all the stuff that cannot be found in a state in the general case.
   - Application-specific events, such that each event defines what happens to the state when it happens. However, some types of events should probably be general.
   - A specific view that presents the output of our program in text format, as specified in the instructions above.
3. A main program that starts a DEDS with an appropriate application-specific start state, view, and event set.

The designers quickly write down the main ideas of the design on a napkin and hand it to you (Figure 1 on the next page). The sketch lacks many details such as methods, packages, and other trivialities. When you point out that they also probably did not even specify all the connections between different classes, their only response is: *That is your job!*

However, they insist that the general parts of the program (the green parts) should not need to know anything about the specific parts of the program (the blue parts). In fact, it should be possible to replace the specific parts with other classes that simulate, for example, a tennis match, a space mission, or a nuclear plant without changing the general parts. That is all the information you get from them before they tell you to leave, you slowly start walking away, and you hear them order yet another cake.

6 Presenting the lab

Before presenting the lab, make sure that

- the program works as expected (try to run simulations with the same parameters as in the example above as well as with other parameters);
- you have written clear *Javadoc* for all classes and public methods; you should also write short package descriptions and put them in files `package.html` in the corresponding source directory;
- you understand both the design and the code (even the part written by your lab partners).

```
Fast machines: 2
Slow machines: 2
Fast distribution: (2,8, 4,6)
Slow distribution: (3,5, 6,7)
Exponential distribution with lambda = 2,0
Seed = 1234
Max Queue size: 5
```

Time	Fast	Slow	Id	Event	IdleTime	QueueTime	QueueSize	Rejected
0,00	2	2	-	Start	0,00	0,00	0	0
0,22	2	2	0	Arrive	0,87	0,00	0	0
0,24	1	2	1	Arrive	0,95	0,00	0	0
0,32	0	2	2	Arrive	1,10	0,00	0	0
0,71	0	1	3	Arrive	1,49	0,00	0	0
1,26	0	0	4	Arrive	1,49	0,00	0	0
2,05	0	0	5	Arrive	1,49	0,80	1	0
2,58	0	0	6	Arrive	1,49	1,85	2	0
2,82	0	0	7	Arrive	1,49	2,58	3	0
2,85	0	0	8	Arrive	1,49	2,69	4	0
4,18	0	0	0	Leave	1,49	9,35	5	0
4,76	0	0	1	Leave	1,49	11,65	4	0
4,92	0	0	9	Arrive	1,49	12,14	3	0
5,37	0	0	10	Arrive	1,49	13,94	4	0
5,89	0	0	2	Leave	1,49	16,54	5	0
6,70	0	0	11	Arrive	1,49	19,78	4	0
7,25	0	0	3	Leave	1,49	22,55	5	0
8,20	0	0	12	Arrive	1,49	26,35	4	0
8,38	0	0	5	Leave	1,49	27,24	5	0
8,53	0	0	4	Leave	1,49	27,82	4	0
8,92	0	0	13	Arrive	1,49	28,99	3	0
9,05	0	0	14	Arrive	1,49	29,54	4	0
9,72	0	0	15	Arrive	1,49	32,88	5	0
10,33	0	0	16	Arrive	1,49	35,95	5	1
10,80	0	0	17	Arrive	1,49	38,28	5	2
10,98	0	0	18	Arrive	1,49	39,15	5	3
11,69	0	0	9	Leave	1,49	42,73	5	4
11,78	0	0	8	Leave	1,49	43,10	4	4
11,94	0	0	19	Arrive	1,49	43,56	3	4
12,13	0	0	6	Leave	1,49	44,35	4	4
12,22	0	0	7	Leave	1,49	44,61	3	4
12,43	0	0	20	Arrive	1,49	45,03	2	4
14,06	0	0	21	Arrive	1,49	49,92	3	4
14,39	0	0	22	Arrive	1,49	51,22	4	4
14,44	0	0	23	Arrive	1,49	51,49	5	4
14,58	0	0	24	Arrive	1,49	52,19	5	5
15,00	0	0	-	Stop	1,49	54,29	5	6

```
Total idle machine time: 1,49
Total queueing time: 54,29
Mean queueing time: 2,86
Rejected cars: 6
```

Figure 1: Example output one.

```
Fast machines: 2
Slow machines: 4
Fast distribution: (2,8, 5,6)
Slow distribution: (4,5, 6,7)
Exponential distribution with lambda = 1,5
Seed = 1234
Max Queue size: 7
```

Time	Fast	Slow	Id	Event	IdleTime	QueueTime	QueueSize	Rejected
0,00	2	4	-	Start	0,00	0,00	0	0
0,29	2	4	0	Arrive	1,74	0,00	0	0
0,32	1	4	1	Arrive	1,91	0,00	0	0
0,43	0	4	2	Arrive	2,32	0,00	0	0
0,95	0	3	3	Arrive	3,88	0,00	0	0
1,67	0	2	4	Arrive	5,34	0,00	0	0
2,73	0	1	5	Arrive	6,40	0,00	0	0
3,44	0	0	6	Arrive	6,40	0,00	0	0
3,76	0	0	7	Arrive	6,40	0,32	1	0
3,80	0	0	8	Arrive	6,40	0,39	2	0
4,90	0	0	0	Leave	6,40	3,70	3	0
5,79	0	0	1	Leave	6,40	5,48	2	0
6,35	0	0	2	Leave	6,40	6,04	1	0
6,56	0	0	9	Arrive	6,40	6,04	0	0
7,16	0	0	10	Arrive	6,40	6,64	1	0
7,54	0	0	3	Leave	6,40	7,40	2	0
8,06	0	0	4	Leave	6,40	7,92	1	0
8,24	0	0	5	Leave	6,40	7,92	0	0
8,93	0	1	11	Arrive	7,09	7,92	0	0
9,87	0	0	7	Leave	7,09	7,92	0	0
10,10	1	0	6	Leave	7,32	7,92	0	0
10,94	2	0	12	Arrive	8,99	7,92	0	0
11,59	1	0	8	Leave	9,64	7,92	0	0
11,89	1	1	13	Arrive	10,24	7,92	0	0
12,07	0	1	14	Arrive	10,42	7,92	0	0
12,49	0	0	9	Leave	10,42	7,92	0	0
12,96	0	1	15	Arrive	10,89	7,92	0	0
13,32	0	0	10	Leave	10,89	7,92	0	0
13,78	0	1	16	Arrive	11,35	7,92	0	0
14,40	0	0	17	Arrive	11,35	7,92	0	0
14,63	0	0	18	Arrive	11,35	8,15	1	0
14,68	0	0	12	Leave	11,35	8,24	2	0
14,79	0	0	11	Leave	11,35	8,35	1	0
15,00	0	0	-	Stop	11,35	8,35	0	0

```
Total idle machine time: 11,35
Total queueing time: 8,35
Mean queueing time: 0,44
Rejected cars: 0
```

Figure 2: Example output two.

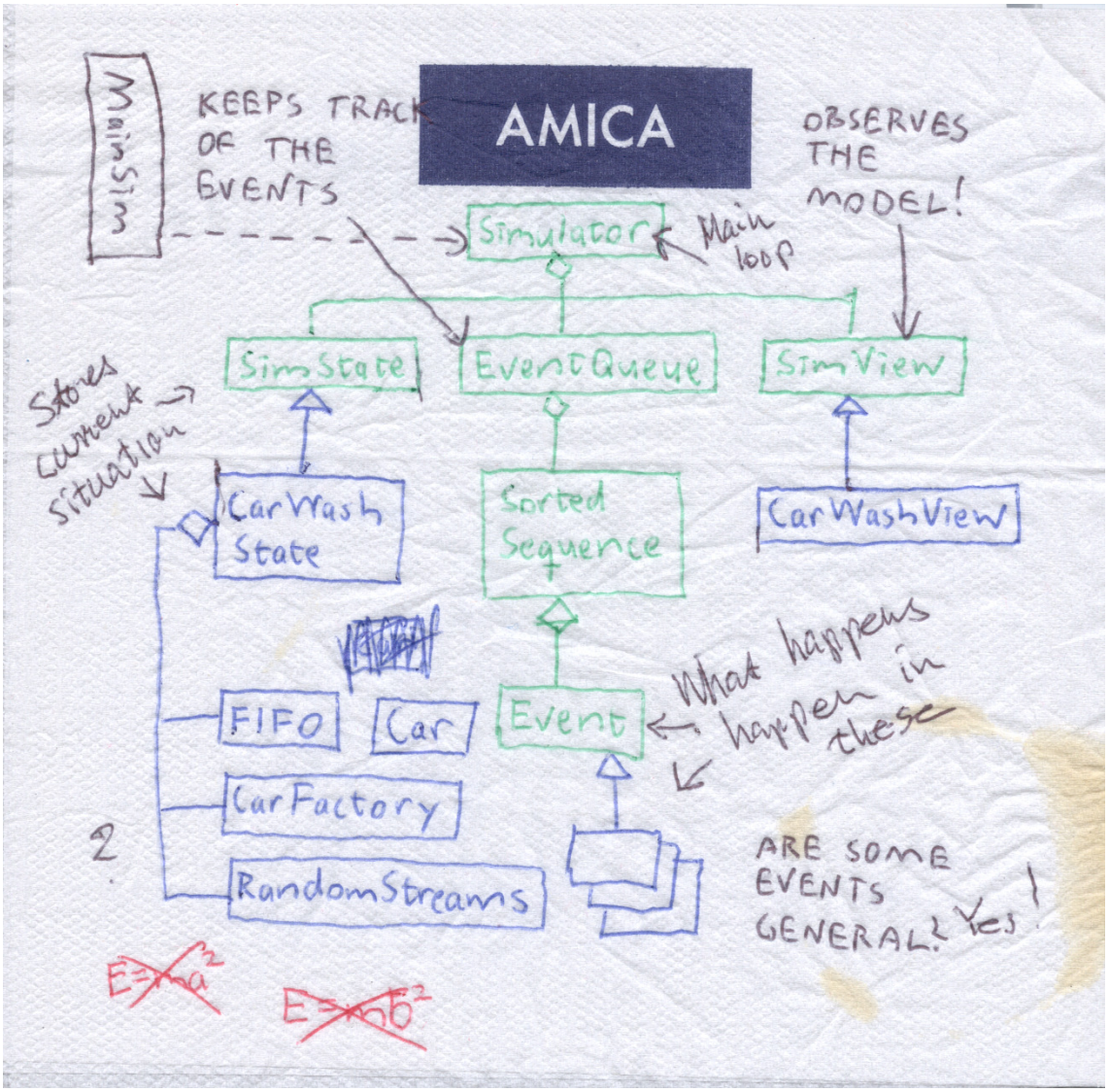

Figure 3: Simulator design: main ideas.