Discrete Event Simulations with J-Sim

Jaroslav Kačer ¹
jkacer@kiv.zcu.cz

Abstract

This paper describes J-Sim, a Java library for discrete-time process-oriented simulation. J-Sim is a fully portable successor to C-Sim, an already existing library written in C. The concepts used in both libraries are inherited from the Simula language. In this paper, the theoretical background, basic principles of implementation, and a simple example of use are presented.

1 Introduction

J-Sim is a software package whose primary goal is to provide an easy way for simulations of discrete event systems in Java. It is strongly inspired by Simula, the first widely used simulation language, developed in the 1960s by Ole-Johan Dahl and Kristen Nygaard. (More information about the history and principles of Simula can be found in [4].) J-Sim offers all concepts known from Simula, including the possibility to model networks consisting of active stations serving a passive flow of customers. Stations are usually composed of two parts: a queue and a server, managing the queue. However, a different approach can be taken: the customers can be represented by active objects, interacting with passive stations. J-Sim is completely independent of any of the two models, it is only a tool allowing a specific model to be described and simulated using the Java language. Moreover, J-Sim is not limited to queueing networks modelling, it can be used for any kind of simulation having discrete-time character.

2 The Simulation World Description

A simulation model can contain a various number of independent active processes. Every process has its own pre-programmed life which can be divided into parts. All processes within the same simulation model share the same time, called simulation time. Its value is equal to zero before the simulation starts and can only be increased during its progress. One part of a process’ life is executed at one exact point of simulation time which does not change during the execution. All parts of all processes’ lives are merged together and arranged according to the value of their simulation time.

¹University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering, Univerzitní 8, 30614 Plzeň, Czech Republic
The execution of the simulation is divided into steps. One step corresponds to the execution of one selected process’ part. The simulation time changes discretely between two consequent simulation steps. The execution is fully under control of the currently executed process, i.e. no other process can interrupt or postpone the execution.

All processes share a calendar where events are stored. An event is an object holding information about a process’ life part; this information contains the process’ identification and a value of simulation time at which the life part is scheduled.

In order to divide their lives into parts, processes use reactivation routines which are able to establish reactivation points in the code of their lives. Two kinds of reactivation routines and reactivation points can be distinguished:

1. A passivating routine (passivate) terminates the current simulation step without adding any new event to the calendar; therefore, the process will not be activated anymore unless another process activates it explicitly.

2. A temporarily passivating routine (hold(Δt)) terminates the current simulation step and adds a new event to the calendar. This causes that the process will be automatically restarted in the future, after Δt time units.

3 Design Decisions

The main goal of J-Sim was to provide a modern, easy-to-learn, and easy-to-use alternative to C-Sim (see [2] and [3]), existing since 1995. C-Sim is written in ANSI C and therefore is not object oriented. The user must use some special constructs (macros) to define a process’ life or an element of a queue and he must be aware of some unusual features, e.g. access to variables of a process. Process switching is implemented using long jumps\(^2\).

J-Sim uses Java threads for implementation of processes (one thread per process) and synchronization routines \texttt{wait()} and \texttt{notify()} to control their activity. The built-in support for concurrent programming was one of the most important arguments why Java was selected as implementation language, together with its platform independence and object orientation.

3.1 Similar tools available

A similar approach can be found in Desmo-J (see [5]), developed at the University of Hamburg and based on the DEMOS simulation toolkit. Another example of process-oriented simulation library based on Java threads is simjava ([6]), where a simulation is a collection of entities (running in parallel), connected by ports and communicating with \footnote{\texttt{setjmp()} and \texttt{longjmp()} functions}
each other by sending events. *sim_tool* (see [7]), a package for simulation of distributed systems, also uses active objects internally based on Java threads.

The main advantage of J-Sim is the compatibility with C-Sim, necessary for education purposes. Despite its relative simplicity (when compared to other tools), it can be used for serious research or industrial projects, as proved in case of C-Sim\(^3\).

4 J-Sim Core Classes

All J-Sim classes (except for GUI classes) are located in the package named *cz.zcu-fav.kiv.jsim*. It is necessary to import them at the beginning of every program using J-Sim. In this article, only the most important of them will be presented.

4.1 The JSimSimulation Class

*JSimSimulation* instances represent theoretical simulation models where various number of processes and queues can be inserted. A calendar (instance of *JSimCalendar*) is owned by every simulation object, where events created by simulation’s processes are inserted. During one simulation step, exactly one event is interpreted and destroyed afterwards.

To the user, the simulation object offers the possibility to execute one simulation step by providing the `step()` method. During the execution, the thread calling this method (usually the main thread of the application) is suspended and it is reactivated when the step finishes. Therefore, there is always one running thread only, never more.

4.2 The JSimProcess Class

The *JSimProcess* class is a ‘template’ for user processes. A method, called `life()`, is introduced in *JSimProcess*, which contains code representing behavior of a process. This method is initially empty and should be overwritten in user’s subclasses.

There are four principal methods which can be used for process scheduling and switching: `passivate()`, `hold()`, `activate()`, and `cancel()`.

1. `passivate()` implements the passivating routine described above.
2. `hold()` implements the temporarily passivating routine described above.
3. The `activate()` method inserts a new event into the calendar and assures thus that the calling process will get control in future. The method takes one parameter: absolute simulation time of activation.
4. The `cancel()` method deletes all process’ events from the calendar. If the process is passive, it will not be woken up anymore unless activated again by another process.

\(^3\)C-Sim is the simulation tool used in the FIT (Fault Injection for TTA) project.
4.3 The JSimHead and JSimLink Classes

The JSimHead class is an equivalent of Simula’s HEAD. It represents the head of a queue where objects of various types can be inserted. However, the class does not provide any methods for insertion or removal of data elements. Instead, the data to be inserted into a queue must be wrapped by an instance of JSimLink, J-Sim’s equivalent of LINK, which is able to insert/remove itself into/from a queue. JSimHead’s useful functions are: empty(), cardinal(), first(), last(), and clear() already known from Simula, plus statistics functions getLw() and getTw(), returning the average length $L_W$ of the queue and the average waiting time $T_W$, respectively.

An instance of the JSimLink class can be inserted at most into one queue, using one of the following methods: into(), follow(), and precede(). The first one takes a queue as its argument while the others use another element, already present in a queue, to insert the caller in the same queue, either before or after the argument. A JSimLink object can remove itself from a queue by invoking its out() method.

5 An Example of Use

Let’s show the possibilities of J-Sim on an example of a simple queueing network, depicted in figure 1. Complete source texts are placed in directory Examples/08_Het QueueingNetworks of the distribution archive.

![Figure 1: An Open Queueing Network](image)

The network contains two servers, each of which has a FIFO queue where transactions waiting to be serviced are put. The transactions (coming from two independent sources) enter the system at two input points and may leave it at two output points, after being serviced. We assume exponentially distributed random arrival time in input streams of transactions and exponentially distributed random service time of both servers. The corresponding parameters of the simulation model are then as follows: $\lambda_i$ is the mean
frequency of the $i^{th}$ input stream (and the parameter of the exponential distribution of arrival time), $\mu_i$ is the parameter of the exponential distribution of the $i^{th}$ server’s service time, $p_i$ is the probability of transaction departure after being served in node $i$ (and with complementary value $1 - p_i$, the transaction passes into node $3 - i$).

We choose the classic approach to construct the model of the network – the servers and the sources of transactions will be active objects while the transaction will be passive.

5.1 Transactions

A transaction is a simple passive object, holding no data except for the time of its creation. See file Transaction.java for details.

5.2 Sources

Being an active object, the source must be inherited from JSimProcess. It is assigned a queue where it stores the transactions generated during its life. See file Generator.java for complete source text.

```java
public class Generator extends JSimProcess { // fields, constructor, ...
protected void life() { // ...
  while (true) {
    link = new JSimLink(new Transaction(myParent.getCurrentTime()));
    link.into(queue);
    if (queue.getServer().isIdle())
      queue.getServer().activate(myParent.getCurrentTime());
      hold(JSimSystem.negExp(lambda)); // ...
  })
}
```

5.3 Servers

Every server has a queue that it takes transactions from. If the queue is empty, the server passivates itself and is restarted later when a transaction is inserted into its queue. After a transaction is taken out, the server processes it (simulated by hold) and puts it into the other queue or throws away. The number of transactions (counter) and the time spent by them in the system (transTq) is registered for every server. The mean response time of the whole network can be therefore easily computed. See Server.java for more details.

```java
public class Server extends JSimProcess { // fields, constructor, ...
protected void life() { // ...
  while (true) {
    if (queueIn.empty()) passivate();
    else {
      hold(JSimSystem.negExp(mu));
      link = queueIn.first();
      if (JSimSystem.uniform(0.0, 1.0) > p) { // throw away
        t = (Transaction) link.getData(); counter++;
The simulation can be executed step-by-step when its `step()` method is repeatedly invoked, e.g. in a `for` cycle. The results are almost identical with the results of the C-Sim version example. They vary according to the $\lambda_i$, $\mu_i$, and $p_i$ values.

## Conclusions

In this article, some basic facts about J-Sim have been presented, including its theoretical background. Being written in Java, a popular and easy-to-learn language, J-Sim should become at least as spread as C-Sim, its predecessor. In the distribution package, source texts, compiled classes, documentation and many examples are included. Today, J-Sim is a fully functional library which has been tested on the examples included in the package. J-Sim is available for free at [1].

## References


## Acknowledgement

This research was supported by the grant of the Ministry of Education of the Czech Republic, No. MSM-235200005 – *Information Systems and Technologies.*