SIMULATING ACTIVITY NETWORKS IN JAVA

Marta Rebello[1], Zita Fernandes[1], Carlos F. Bispo[1]

[1]Instituto de Sistemas e Robótica, Instituto Superior Técnico,
Av. Rovisco Pais, 1049-001 Lisboa, Portugal.

cfb@isr.ist.utl.pt (Carlos F. Bispo)

Abstract

In order to evaluate the performance of several scheduling policies for multiclass queuing networks we felt the need to develop a general purpose software package where different networks and policies can be tried, either from a set of pre-defined systems and policies or for user-defined systems and policies. In this paper we present a package to perform discrete event simulation of a manufacturing system with different configurations specified by the user, according to the framework of activity networks as introduced by Harrison in [1]. The package uses an object oriented modeling technique. The language in question is Java.
The architecture of the system to be simulated is described in file format, where the resources, routing patterns for customers and other parameters are specified. The user may choose the decision policies from a library of methods or can add new policies to this library. The modeling flexibility permitted includes standard multiclass queuing networks to be modeled, where there are a few types of customers, each characterized by a unique routing, at each stage of the routing customers wait in a specific buffer for the next service, and each server works on a customer at a time. The activity networks allow other configurations, where a given service may take more than a customer from different buffers. Some preliminary studies on the performance evaluation of distributed scheduling policies for queuing networks are presented as a means to provide insights on the type of utilization this package can have.

Keywords: activity networks, java simulation package, scheduling policies, performance evaluation

Presenting Author's biography

Carlos F. Bispo obtained the degrees of "Licenciatura" and MSc. on Electrical and Computer Engineering at the Instituto Superior Técnico, the engineering school of the Technical University of Lisbon, Portugal in 1985 and 1988, respectively. He obtained the degrees of MSc. and PhD on Industrial Administration, in 1993 and 1997, at the Carnegie Mellon University, Pittsburgh, USA.
He is currently a tenured Assistant Professor at the Department of Electrical and Computer Engineering of the IST and conducts his research activities at the Instituto de Sistemas e Robótica.
His main research interests are in the area of Operations Management, with particular emphasis on Scheduling for Queuing Networks, and Inventory Control.

ISBN 978-3-901608-32-2

1. Introduction

In order to evaluate the performance of several scheduling policies for multiclass queuing networks we felt the need to develop a general purpose software package where different networks and policies can be tried, either from a set of pre-defined systems and policies or for user-defined systems and policies.
The simulation starts with specifications defined by the user that allows the creation of virtual elements, according to object oriented programming, which will be used to construct the modeled system. The initial specifications will map the input variables, the objects and the attributes for the objects. The existent templates allow the simulation of the general activity networks first introduced by Harrison in [1].
The initial specifications of the input are according to the minimal elements needed to the construction of the network [1]. This network will be constituted by two kinds of objects: machines or servers and buffers (queues). The dynamics between the elements will be made according to activities that will obey to scheduling policies. These policies, according to [2], may be such that they reduce the mean and variance of cycle-time, but they can also address other performance concerns.
The events will result from the movement of materials such as packets, customers, etc. Each server uses several resources to produce outputs of materials that may be of various (possibly different) kinds. These materials can be processed by several servers, can be split up, or combined with other kinds of materials before a final output is produced.
In what follows we will briefly describe the nature of the networks we will be interested on simulating, describe the main events that are being considered, and how the scheduling policies will be addressed. Next section will discuss the structure of the simulator by means of the Unified Modeling Language (UML). Section 3 briefly describes how data will be collected and processed. Section 4 will present a numeric study that serves as an illustration on the utilization intended for the package. Finally, we will present some concluding remarks.

1.1 Processing networks

In this section we will describe the activity network. The model of the network was made following the processing network developed by Harrison in [1]. We assume to have l servers (represented by circles in Figure 1) and j types of activities (represented by the broken arrows). The model has i input buffers (that receive material from the outside) with an average arrival rate $ë_i$ and service (intermediate) buffer that receive materials after processed. Each activity, j, uses a particular server with average service rate $ì_j$ materials by unit time, processes materials from one or more given buffers and places them in buffers after completion of service. We have q_k units of each server, expressed as the number of available parallel servers per time unit.
Again according to Harrison, we consider that R_{ij} is the average amount of material of buffer i consumed per unit by activity j. All components of R_{ij} are assumed to be integer positive or negative numbers. They are negative when a buffer receives the outcome of the activity and positive when the buffer feeds the activity.
We also assume that A_{kj} is the average rate at witch activity j consumes the capacity of resource k.

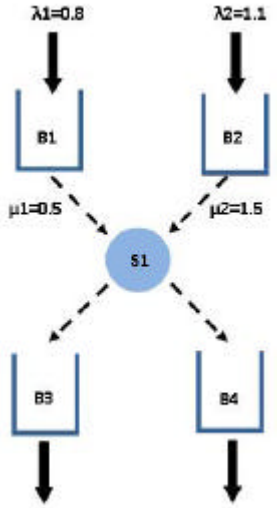

Figure 1. Example of a simple activity network.

In this simple case (Figure 1) we only have one server able to process 2 types of activities from and to 4 buffers. We can see that knowing only the user-specified parameters R, A, $ë$, and q, plus the details of the failure and repair processes, we are able to model and simulate the network during a specified time interval. Using still Figure 1, a positive value of R_{11}=2 means that activity 1 will use 2 units of material from buffer 1 and a negative value of R_{13}=-1 means that the same activity will produce only 1 unit of material to buffer 3. The existing server will process activity 1 at an average rate of 0.5 per unit time and activity 2 at an average rate of 1.5 per unit time. The input to buffers 1 and 2 (both input buffers) will have average rates of 0.8 and 1.1 per time unit.

1.2 Network dynamics

The dynamics of the network is based on the events. These events can be of various types: arrivals, end of services, server failures and end of repair (of the servers).
- Arrival: When something (someone) arrives from the outside this material (or customer) is created and placed on a specified buffer. It also verifies if any activity can be initiated according to the policies of the servers. If so, an end of service event will be

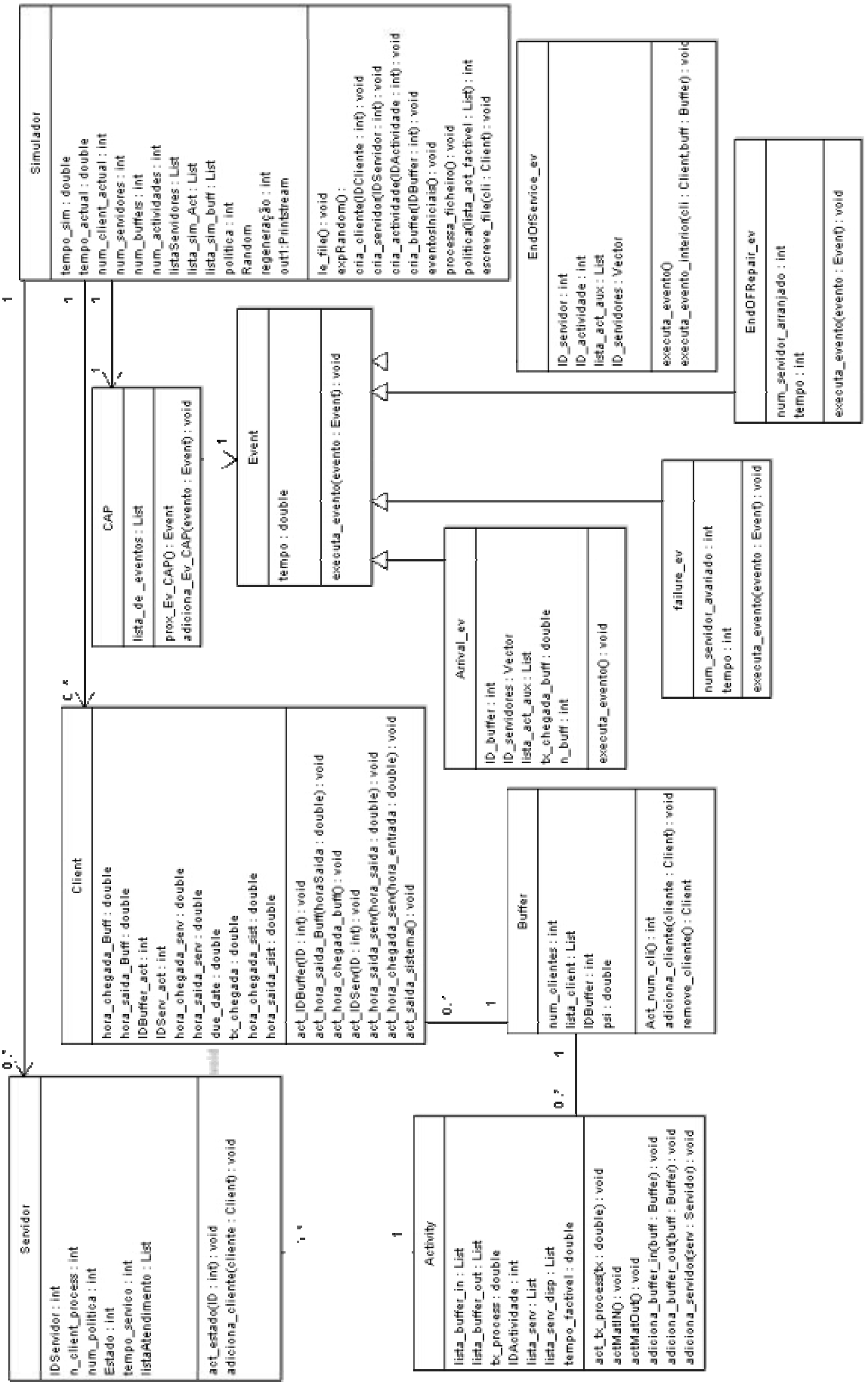

Figure 2. The Unified Modeling Language for the simulator.

created and placed on the CAP – Class that contains the list of pending events.
- End of service: When this type of event happens the material that has been processed is placed on the specified buffer or deleted (witch corresponds to a departure from the network) according to the activity in question. The server in question now has to verify if it is possible to start another activity.
- Failure: The first set for these events is created at the beginning of the simulation. If the server is working when a failure happens there are 3 options: the objects that were being processed are destroyed, the objects that were being processed are placed at the beginning of the buffer and the activity will start over again when the server restarts its activity, or finally the objects remains at the server and the end of service is only delayed. At this point a new event, end of repair, is placed on the CAP.
- End of repair: The server in question is working again and verifies if it is possible to start a new activity or continue the old one. If so, an event end of service is added to the CAP.

1.3 Policies

As we are intending to explore performance improvement for the policies referred on [2], since they have been shown to be more efficient than the usual ones, the first version of the package will include these in the default library. The objective is to use Moreira's controller, [3], to see if there is any possibility for improvement on those.
These will be implemented and the user will be able to select which one he/she wants to use on each server. It will also be possible for the user to create different policies that will be part of the package's heritage as its utilization grows.

2. Object oriented programming

In addition to some of the parameters mentioned above, (R, A, $\ddot{e}$ and q), we have to specify the frequency of failures in the servers, the mean time to repair the server and sometimes a policy requires some additional information as due dates or remaining cycle time. All these parameters are defined in the input file that allows the generation of the objects according to an oriented object modeling processing language.
For each component of the network (servers, buffers and activities) one virtual object will be created from the template defined. According to the template of the input file, the users only have the possibility to change the attribute value of the objects.
If we think in an activity network, and not in the particular case of a queuing system, the servers can also be machines and the clients can be jobs or packages of material. The materials moving or arriving to the network will also have a virtual object associated, with several attributes that will be helpful to collect results, at the end of the simulation.
The easiest way to manage all these objects, which can be very high in number, is to use lists (Linked lists). For this, Java already has a class that allows us to organize the objects as needed, to create all the dependencies of the network that we want to build or, in the case of clients, to remove and add them from the buffer (each buffer may hold multiple clients). For instance in this case, each activity has a list of servers and a list of buffers. Each buffer has a list of clients. These clients are moved according to their type and the policy used by the server. The reason why each activity has a list of servers is quite obvious. Each activity may be processed by one or more servers in parallel. If an arrival occurs to a buffer we need to determine which activity can be processed and if the correspondent server is available. If there are more than one server available, we currently choose one by random. To do differently one would need to specify routing policies, which is an option not yet included in the package. In the case where an arrival to a buffer enables more than one activity, we use the scheduling policy coded for the corresponding server to choose the activity that will be performed.
In the UML (see Fig. 2, above), the arrows symbolize the relations between the different classes (types of objects). These arrows can be bidirectional or unidirectional, depending if the relation works in both ways or not. For example, a buffer has a list of clients, but a client can only belong to one buffer, this is a **one to many** relation, thus bidirectional. That is why we have the multiplicity specified near the arrow – the string "0..*" means zero or more objects of that class. Another example is the simulator. When the simulation starts, we create an instance of the simulator and this simulator has one CAP, but the CAP does not have a simulator. This is an example of a unidirectional relation.
The simulator is the main object of the program [4]. It is from this object that the program creates the network, starts and finishes the simulation, proceeds to the execution of the different events, and also has the methods defining the available policies. Each policy has its own method, so the more policies we want to test, the more methods we have to create.
An event is an occurrence associated with the beginning or ending of an action. It can be an arrival of a client from the outside, a end of service (this means that the client leaves the server and goes to the next queue, or leaves the system, and that the server becomes ready to do another service), a failure (when a server gets broken it can no longer process any activity), and finally an end of failure (when the server becomes functional again).
Each time that an event is generated we need keep it ordered with the other events that have already been

generated. To do this we use another Linked List together with the **Comparable Interface** to ordinate by time these events. This is necessary because we need to make sure that the next event processed by the simulator has the earliest time tag. This is the only case were we need to order any list. For the other lists it only matters the arrival order, so we do not have to change anything.
The events just described are saved in the CAP class.

3. Data collecting

The data resulting of the simulation will be completely processed after the simulation is over. The data corresponding to each object will be saved on a text file during the network simulation. By doing this, we will avoid wasting capacity of processing and only use the computer processor to execute the dynamics of the network, that is the movements of the materials between the different network objects. All the mathematics needed to analyze and conclude about the simulation will be done using the Matlab package.
The simulation period can be determined by three different ways: by reaching the number of unit materials that were processed by the network pre-specified by the user, by a period also defined by the user or by reaching a user specified number of regeneration points. A regeneration point is said to be reached when all the objects of the network, buffers and servers, are empty. That is, when the system is completely empty of customers.

4. Results

In this section we will now present some results obtained using one of the policies developed in our research with the goal of comparing the μc-rule with alternative policies. The μc-rule is known to be optimal in the context of scheduling decisions when two classes of customers arrive to a system composed of a single server and costs are affine on the queue lengths, for a Markov Decision Problem, MDP, known as the Scheduling Problem [5].
The policy is such that the class with the highest score when one multiplies the processing rate, μ, by the cost rate, c, gets priority over the other class.
This means that the non-priority class will only get access to the server when, upon completion of a service the priority queue is empty of customers. Naturally this has a strong influence on the total waiting time for the non-priority class. We question the fairness of such treatment and propose that costs of waiting for service should not be linear functions of the total waiting time. We argue that the marginal "patience" of any given customer to wait another minute should not be independent on the amount of waiting time already incurred.
Therefore, we formulated a MDP where we introduce a quadratic component on the queue length as an extra term of cost. Solving the MDP numerically using Dynamic Programming we arrive at a control policy that then we can simulate to obtain several performance metrics. Our results so far on the MDP solution seem to point in the direction that high priority should be awarded to the non-priority class when its buffer size exceeds some threshold, which is a function of the quadratic parameter for the non-priority class. Thus, we implemented such policy on our simulator and collected data on the average total time in the system and respective variance as a function of the threshold.
According to the criteria described in Section 1 the input matrixes are:

$$A = [1 \quad 1] \qquad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$L = [0.475 \quad 0.475] \qquad Q = [1]$$

$$costs = [2 \quad 1]$$

The first figure (Fig.2) shows the variance of the service and waiting time for the two classes of customers and Figure 3 displays the average of the service and waiting time for the two classes of customers.

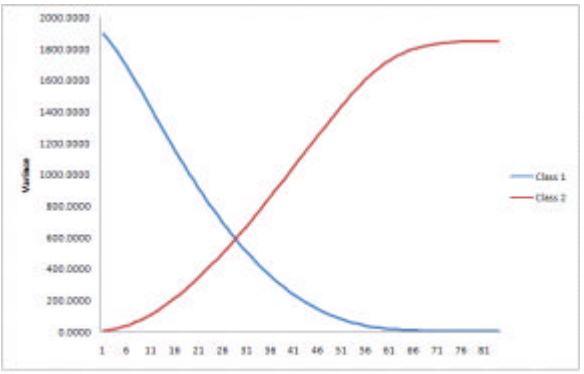

Figure 3. Variance of the average total time

These results were obtained for increasing values of the threshold, k, starting with 1 and ending with 85. As the threshold grows to infinity the behavior converges to the μc-rule. So, we observe that for high values of k the total average time in the system and variance for the non-priority class, Class 2, are both high, when compared to the same values for the priority class, Class 1.
As k drops to 1, we are awarding progressively more priority to Class 2, until we reach the opposed extreme of excessive degradation on performance for Class 1. For intermediate values of k it is possible to balance the quality of service for both classes, in terms of the average total time in the system, but more importantly the variance of the total time in the system.
The results obtained correspond to 1000 regeneration intervals and the 95% confidence intervals were

computed for intermediate values of k, such that the spread around the mean is under 5%.

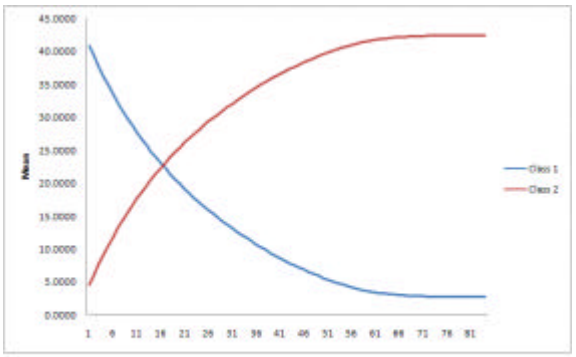

Figure 4. Average total time.

With these results we show one application of many possible and also demonstrate the flexibility of the package on this particular case by modifying it as needed easily. As we change the policy's parameters as many times as we need, we can also add new policies to the simulator, by simply creating a new method that returns the activity to execute next. In addition, we could also have changed the network architecture by modifying the input.
We are in the process of testing more systems and alternative scheduling policies. We will be able to present more extensive data during the presentation.

5. Conclusions

We presented a simulation package, written in JAVA, that will allow general studies on the performance of given scheduling policies for general systems described with the structure of activity networks. Although initially developed for a specific study, a significant effort has been placed on producing a sufficiently general discrete event simulator for networks of processing activities, which include, but are not limited to, queuing networks as a subset.
The modeling flexibility permitted includes standard multiclass queuing networks to be modeled, where there are a few types of customers, each characterized by a unique routing, at each stage of the routing customers wait in a specific buffer for the next service, and each server works on a customer at a time. The activity networks allow other configurations, where a given service may take more than a customer from different buffers.
To illustrate its purpose, some preliminary performance evaluation studies were presented. The studies concern a variation on the classic Scheduling Problem, with the extra concern on reducing variance for the non-priority class.
By the time of the conference we hope to be able to present some preliminary results of the specific study we intend to conduct, as an example of the flexibility and usefulness of the package we are now developing.

6. Acknowledgement

This work was supported by Fundação para a Ciência e a Tecnologia (ISR/IST pluriannual funding) through the POS_Conhecimento Program that includes FEDER funds.

7. Bibliography

[1] Harrison, J.M., Stochastic networks and activity analysis. Y. Suhov, ed. Analytic Methods in Applied Probability. In memory of Fridrik Karpelevich. American Mathematical Society, Providence, RI, 2002.
[2] Lu, S.C.H., Ramaswamy, D., and Kumar, P.R., "Efficient Scheduling Policies to Reduce
Mean and Variance of Cycle Time in Semiconductors Manufacturing Plants" IEEE Trans. Semicond. Manuf, vol. 7, no3, pp 374-398, 1994.
[3] J. Moreira, J.A. and Bispo, C.F., Performance Improvement through Active Idleness, 11th Mediterranean Conference on Control and Automation Rhodes, Greece, June 2003.
[4] Arnold, K., Gosling, J., and Holmes, D.,, "Java Programming Language", Fourth Edition, Addison Wesley.
[5] Cassandras, Christhos G., Discrete Event Systems: Modeling and Performance Analysis, Asken Associates Incorporated Publishers, 1993.

