# Computer Assignment 04: Transfer Line Model

## Concepts

* More experience with Entities and containers
* How to create “Arrays” of containers using Lists of containers
* Events with more than one argument
* IndexedPropertyChange events
* Using “Multiple” Stats to collect statistics on data fired by IndexedPropertyChange events

## Problem Description

Jobs arrive to a production facility consisting of a number of workstations in a line (transfer line). Each job must be processed at every workstation in sequence. Every workstation consists of a number of identical machines. This exercise will implement the transfer line component in Simkit and run a test on a small facility.

In addition to collecting the “usual” statistics at each workcenter, the average total delay in queue and total time in the system per part will be estimated. Finally, Little’s formula will be examined for each workstation.

## Classes

* Part (Entity)
* PartArrivalProcess (extends ArrivalProcess)
* TransferLineComponent (extends SimEntityBase)
* RunTransferLineComponent (main class)

## Part Entity

Create a Part entity (i.e., subclass Entity – see Figure 1). Since each part will be moving through a sequence of workstations, add an attribute called totalDelayInQueue (abbreviated ‘p.D’ in the Event Graph). This attribute is initialized to 0.0. In addition to a getter method, you will need a method to increment the delay in queue each time the part starts service at a new workstation (see Transfer Line Event Graph in Figure 3).



Figure . Part Class

## PartArrivalProcess

The PartArrivalProcess component follows a familiar pattern by extending ArrivalProcess.



Figure . Part Arrival Process Event Graph

This component needs to schedule an Arrival(p, i) event to conform to the Arrival(p, i) event in the transfer line component (Figure 3).

## Transfer Line Event Graph Component

The Transfer Line Component is defined below by its parameters and state variables. Note that some parameters as well as state variables are defined as arrays. This results in some special ways of initializing their values as well as introducing additional setters/getters (for parameters) and additional getters (for state variables).

### Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Abbreviation |
| totalNumberMachines | int[] | ki |
| processingTimeGenerator | RandomVariate[] | tSi |

### State Variables

|  |  |  |  |
| --- | --- | --- | --- |
| State Variable | Type | Initial Value | Abbr |
| numberAvailableMachines | int[] | totalNumberMachines | S |
| queue | List<SortedSet<Part>> | empty | q |
| delayInQueue | double[] | {NaN} | Di |
| timeAtStation | double[] | {NaN} | Wi |
| totalDelayInQueue | double | NaN | DT |
| totalTimeInSystem | double | NaN | WT |

Since introducing generics Java has disallowed instantiating arrays of containers. Therefore, when it is desired to have such an array, a List must be used instead, as shown in the table.

### Event Graph



Figure . Transfer Line Component[[1]](#footnote-1)

## Implementation Notes for Transfer Line Component

* A separate parameter for the number of workstations (n) is not necessary – use totalNumberMachines.length as the number of workstations in the code
* Instantiate the arrays and the queue container in the zero-argument constructor.
* The queue container should be instantiated in the constructor using a for loop as follows:

queue = new ArrayList<SortedSet<Part>>();

for (int station = 0; station < totalNumberServers.length; ++station) {

queue.add(new TreeSet<>());

}

* For each array state variable there should be two getters: one that returns a copy of the array (the easiest way is to return a clone()) and one that returns just one indexed one. For example, the numberAvailableServers state variable should have the following two getters:

public int[] getNumberAvailableMachines() {

return numberAvailableMachines.clone();

}

public int getNumberAvailableMachines(int station) {

return numberAvailableMachines[station];

}

* In a similar manner, there should be two getters for queue (note how the second one returns a deep copy):

public SortedSet<Part> getQueue(int station) {

return new TreeSet<>(queue.get(station));

}

public List<SortedSet<Part>> getQueue() {

List<SortedSet<Part>> queueListCopy = new ArrayList<>();

for (int station = 0; station < queue.size(); ++station) {

queueListCopy.add(getQueue(station));

}

return queueListCopy;

}

* Note also how the getQueue(int station) method is used for the state transition shown above.
* The signature for the Arrival event is (Part part). Be careful how you write the methods. For example, this event is implemented as follows (compare with the Event Graph above):

public void doArrival(Part part, int station) {

part.stampTime();

SortedSet<Part> oldQueue = getQueue(station);

queue.get(station).add(part);

fireIndexedPropertyChange(station, "queue", oldQueue, getQueue(station));

if (numberAvailableMachines[station] > 0) {

waitDelay("StartService", 0.0, Priority.HIGH, station);

}

}

* Note that fireIndexedPropertyChange() is used for the array state variables, but the two non-array state variables will simply use the old firePropertyChange(). Since these are not time-varying state variables, do not use the “old value” when doing so. However, *do* use the old value form for numberAvailableMachines and queue.
* Important: the state transition p.next = p.next + 1 in the Advance(p) event must be implemented as a method in the Part class.

## Listener

The listener diagram for the full model is as follows:



Figure . Listener Diagram

## Main Class

Use the data as shown in the output below for your main class. The sample run will be for 1,000,000 time units on a line consisting of 3 workstations with 5, 4, and 2 servers, respectively, and service times as indicated in the output below.

### Collecting Statistics

Collecting statistics for a numerical array (like numberAvailableServers) and Collections state variables (like queue) are done with two new ‘stats’ classes: MultipleSimpleStatsTimeVarying, MultipleCollectionSizeTimeVaryingStats, and MultipleSimpleStatsTally. As with the ones you have already used, they are instantiated with the name of the state variable they will listen to. For example, to collect the time-varying average number in queue for each workstation, use:

MultipleCollectionSizeTimeVarying numberInQueueStat =

new MultipleCollectionSizeTimeVarying("queue");

transferLine.addPropertyChangeListener("queue", numberInQueueStat)

Similarly, to collect the timeAtStation statistics, use:

MultipleSimpleStatsTally timeAtStationStat =

new MultipleSimpleStatsTally("timeAtStation");

tranferLineComponent.addPropertyChangeListener(“timeAtStation”,

timeAtStationStat);

The other stats objects are defined similarly and listen to the transferLine instance. Each of the “Multiple” stats have getters with the index; for example, to get the average number in the queue at workstation station, use getMean(station). Consult the Javadoc for Simkit for a complete set of methods for these classes.

Estimate the means for two additional states, totalDelayInQueue and totalTimeInSystem, using SimpleStatsTally. Note that the getCount() method of the SimpleStatsTally you use for totalTimeInSystem gives the total number of completed parts.

### Little’s Formula

It turns out that Little’s formula holds (more or less, depending on whether the station is empty or how long the simulation is run) at each workstation. Use your model to verify this (see the output below). You can use the total number of arrivals to the system to estimate the arrival rate.

## Output

PartArrivalProcess.1

interarrivalTimeGenerator = Exponential (1.700)

TransferLine.2

processingTimes[0] = Gamma (3.200, 2.300)

processingTimes[1] = Uniform (4.500, 6.700)

processingTimes[2] = Exponential (3.000)

totalNumberMachines[0] = 5

totalNumberMachines[1] = 4

totalNumberMachines[2] = 2

Simulation ended at time 500,000.00

Number arrivals: 294,636

Number Completed: 294,621

Avg Avg # Avg Delay Avg time

station util in Queue in Queue At Station

0 0.8687 3.0241 5.1319 12.5025

1 0.8250 0.7587 1.2876 6.8883

2 0.8834 5.0118 8.5054 11.5040

Using Little's Formula

Avg delay Avg time

station inQueue at Station

0 5.1320 12.5025

1 1.2876 6.8880

2 8.5050 11.5034

Avg time in system: 30.8946 (Using Little: 30.8940)

Avg total delay in queue: 14.9248 (Using Little: 14.9245)

## Deliverables

Push you code to Gitlab by COB of the due date.

1. [↑](#footnote-ref-1)