ENGSCI 355 Labs

Thomas Adams

2024-08-29

Table of contents

Preface

These are an online version of the Labs for ENGSCI 355. The topics covered are: a hands-on simulation of a manufacturing process; conceptual modelling using HCCM; implementing HCCM models in Jaamsim; and missing data imputaton.

Part I

Practical Lab

1 Operations System in Practice

The goal of this lab is to give you some hands-on experience with an operations system, the type of system that we will be focussing on simulating. Hopefully this will give you some idea of what is needed to simulate a system in terms of:

- the components of the system and how they interact with each other (entities and their behaviour);
- the type and amount of information/data that is needed, both for activity durations and control policies;
- the types of experiments that can be performed and how the system can be redesigned.

1.1 Making Paper Cars

The system that we will use as an example is making a car out of paper. You will each be given a piece of paper with the net of paper car on it as in Figure 1.1.

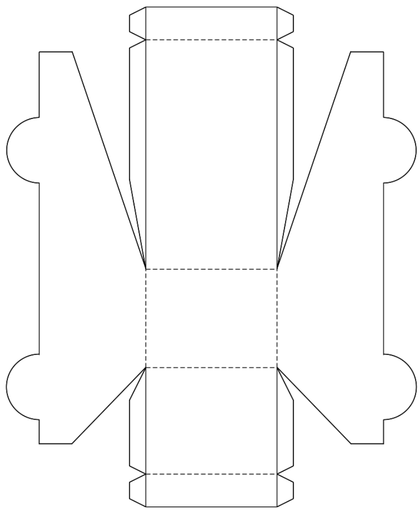

Figure 1.1: The Net Used to Make Paper Cars

You will also get a pair of scissors, some tape, and blank pieces of paper. To make the car:

1. Trace the net onto a new piece of paper.
2. Cut the new net out.
3. Fold the paper and tape the edges shut placing the tabs on the inside.

Figure 1.2 shows an example of a completed car.

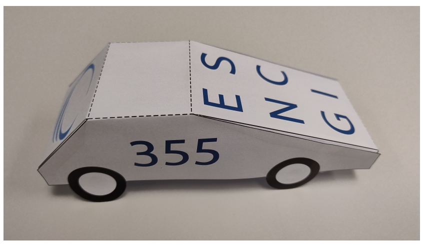

Figure 1.2: A Completed Car

First everyone should make one car by themselves. Once you have, show one of the instructors to get signed off. Then, discuss with you group how you can work together to make paper cars. You might want to experiment with different setups/policies and try making a few cars to see how long it takes and gather some data.

There will be a competition to see which group can make the most cars in 10 minutes. Before the time starts each group must submit an estimate of how many cars they believe they will be able to make. The score for each group will then be comprised of the following elements:

- 1 point for each car completed up to and including the estimated number.
- 0.25 points for each car completed above the estimated number.
- -0.75 points for each car not completed in the estimated number.

Additionally, the following rules must be followed:

1. Each car must be traced and cut individually.
2. Cars must be the same shape as the original template, including tabs.
3. You can have as many stencils as you like.
4. All final cars must have started as a blank, unfolded piece of paper.
5. You may not have any pre-cut tape or nets.
6. All cars must have been made only by members of your group.
7. All cars must be folded and taped neatly to count. The lecturer has final say on whether a car meets the required neatness.

1.2 Reflections

Now that you have attempted to make as many cars as you can you may wish to reflect on the process by asking yourself the following questions:

- Did your group have any traced/cut out cars left at the end?
- What was the bottleneck/slowest part of the system?
- Did you collect any data/do any experiments? If so, did they help? Would you do more/different ones now?
- What would you do differently next time?

The process that we considered was relatively simple. Cinsider how would your group's strategy change if any of the following additional conditions were added:

- Blank pieces of paper for you to trace onto only become available one at a time every two minutes;
- You have to make different styles of cars on demand;
- There is a limit to how many traced nets/cut pieces of tape you can have at any point (buffer limit);
- Each time a pair of scissors is stopped being used there is a cooldown time of 1 minute.

Part II

Jaamsim Labs

2 Using Traces and Scenarios

In this lab we will modify the simulation developed in the previous lab to run off of a pre-generated data trace that contains information about each patient. We will also explore how Jaamsim's built-in scenario indices can be used to run experiments where the values of the simulation's inputs are changed and use an EventLogger to log all events that an entity participates in. Finally we will package the simulation (Jaamsim and the custom Java code) as a .jar file, so that the simulation can be run easily from the command line on all major operating systems.

We are not considering any changes to the system, so the conceptual model is the same as for the previous lab.

2.1 Jaamsim Model

To run the simulation from a data trace we need to make some changes to the Jaamsim model. Once again create a new folder called **RC3** and copy your .cfg file (and the .png files so that the graphics work) from the previous lab folder into this folder and rename it to **radiology_lab_trace.cfg**. First, download the **RC_50_week_data.txt** file from Canvas. This file contains 50 weeks of data of patients at the radiology clinic including: the time the patient arrived, the priority of the patient, the time the patient took to check in, and the time the patient took to have their scan.

Before we load the data in we will first change the starting date of the simulation, which defaults to 1970, to instead be 2024, so that the data read from the file is interpreted correctly. To do this go to the Simulation object and under the Options tab enter **2024-01-01** for the StartDate.

To use the data in Jaamsim we use a **FileToMatrix** object found in the Basic Objects palette. Create a FileToMatrixObject, rename it **PatientData**, and select the **RC_50_-week_data.txt** file as the DataFile.

We can now access the data in the file by using the **Value** output of the PatientData object. The first place we will use this data is in the PatientArrival object, so that patients arrive according to the data in the file, rather than the distribution used previously. We first create two CustomOutputs (under the options tab) on the PatientArrival object to make accessing the data easier. CustomOutputs are similar to attributes but they can be expressions (formulas) and are re-calculated at each time step in the simulation. The two outputs we create will correspond to the data for the patient that has just arrived (thisPatientData) and the patient that is going to arrive next (nextPatientData). We need both of these so that we can calculate the appropriate interarrival time between the patients.

Once we have created these outputs we use them in the InterArrivalTime, and AssignmentList of the PatientArrival.

Table 2.1: Update PatientArrival

Object	Keyword	Value
PatientArrival	CustomOutputList	{ thisPatientData '[PatientData].Value(this.NumberAdded + 1)' }
		{ nextPatientData '[PatientData].Value(this.NumberAdded + 2)' }
	FirstArrivalTime	[PatientData].Value(2)(2)
	InterArrivalTime	'this.nextPatientData(2) - this.thisPatientData(2)'
	AssignmentList	{ 'this.obj.priority= this.thisPatientData(3)' }
		{ 'this.obj.checkInTime= this.thisPatientData(4)' }
		{ 'this.obj.scanTime= this.thisPatientData(5)' }

Note that in the AssignmentList we are assigning values from the data file to attributes on the patient entity for priority, check in time, and scan time. We will use these attributes later to determine how long those activities take (the priority attribute is already used in the PriorityBranch).

To avoid getting an error these attributes need to be added to the PatientEntity object. So update the AttributeDefinitionList of the PatientEntity to include checkInTime and scanTime as well as the current priority, all with a default of 0.

We now need to use the checkInTime and scanTime attributes to determine how long the check ins and scans take. Set the Duration of the CheckIn activity to **this.CurrentParti-cipants(1).checkInTime * 1[min]**. this.CurrentParticipants refers to the group of entities that have just started the activity (for check in this is a patient and a receptionist), and we use the index 1 as the patient comes first, then we access the checkInTime attribute. We then need to multiply this by 1[min] to convert the number into a time, and use minutes as the attribute is in minutes.

Similarly for the Scan activity set the Duration to **this.CurrentParticipants(1).scanTime * 1[h]**, note that here we use 1[h] as the attribute is in hours.

Now, suppose we are interested in the time that patients spend waiting for check in and for the scan. We can't use the current PatientLogger as it only records the total time that patients are in the system for. We could add attributes for each time that we are interested in, and assign the value when the entity gets to the relevant stage, and then use the PatientLogger to log these attributes. We can instead use an EventLogger from the HCCM palette. The EventLogger records the time that an entity starts each of the activities that it participates in. So, create an EventLogger and call it PatientEventLogger.

Then, to get the events recorded go to the PatientLeave object and under the HCCM tab enter PatientEventLogger for the EventLogger keyword. Now any entities that are sent to the patient leave will have the start times of any activities that they participated in recorded.

We will now configure the Simulation object to run one long replication for several scenarios. Under the Key Inputs tab enter **50w** for the **RunDuration**, this will make the simulation run for 50 weeks. We have to run one 50 week replication rather than 50 one week replications as Jaamsim cannot read in a new file when each replication starts.

We want to try out four scenarios with either three or four CT machines, and either one or two receptionists. As there are two factors we are changing we use a ScenarioIndex with two numbers, the first indexes the scenarios relating to the number of CT Machines, and the second those related to the number of receptionists.

Since there are two options for the first index and two for the second we enter **2 2** for the **ScenarioIndexDefinitionList** under the **MultipleRuns** tab of the **Simulation** object. We will start from scenario 1 and end at scenario 2 in both the indices so StartingScenarioNumber is **1-1** and EndingScenarioNumber is **2-2**. We are going to run just one long replication for each scenario so set the NumberOfReplications to 1.

Now Jaamsim will run 4 scenarios, but we need to make it so that the number of CT Machines and Receptionists actually changes in each of the scenarios. For the CT Machines we set the **MaxNumber** on the **CTMachineArrival** to **[Simulation].ScenarioIndex(1) + 2**, which gets the value of the first scenario index and adds 2 to it. For the Receptionists we can set the **MaxNumber** on the **ReceptionistArrival** to **[Simulation].ScenarioIndex(2)**, in this case we don't need to add one as the scenario index is the same as the number of receptionists we want to use.

Download and run the **RadiologyLabTraceAnalysis.R** file, you will have to update the directory that it reads the data from and the name of the data file used. The script splits each replication into 50 batches, each one week long, and calculates the mean across the batches and the four scenarios of the 90th percentile waiting time for both check in and scan within each of batch. No warm-up period is used, so this assumes that being empty and idle is a typical state of the system. Splitting into batches by week assumes that each week is not correlated to the preceding and following weeks. You should get the following output:

Part III

Conceptual Models

3 Making Paper Cars Conceptual Model

3.1 Understanding of the Problem Situation

In the box below write a problem description for making paper cars, think about what you are trying to solve/discover by simulating this activity. You may want to look at Chapter 1 again to remind yourself about the process.

3.2 Modelling Objectives

In the box below write the modelling objectives for making paper cars, i.e., what are you trying to discover using simulation?

3.3 General Objectives

In the box below write the general objectives for making paper cars, i.e., what are some of the general properties you'd like your simulation to have?

3.4 Defining Output Responses

In the box below write the output responses for making paper cars, i.e., what are you going to measure to determine the performance of the system?

3.5 Defining Input Factors

In the box below write the input factors for making paper cars, i.e., what are you going to change to achieve the modelling objectives?

3.6 Identifying Entities

In the box below list the entities for making paper cars.

3.7 Drawing Behavioural Paths

The activity diagrams for the pencil & template, and scissors are given below in Figures 3.1, and 3.2.

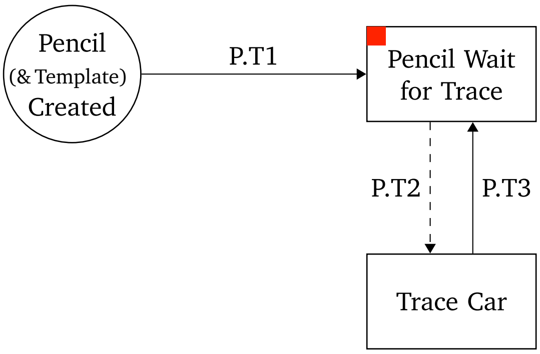

Figure 3.1: Pencil Activity Diagram

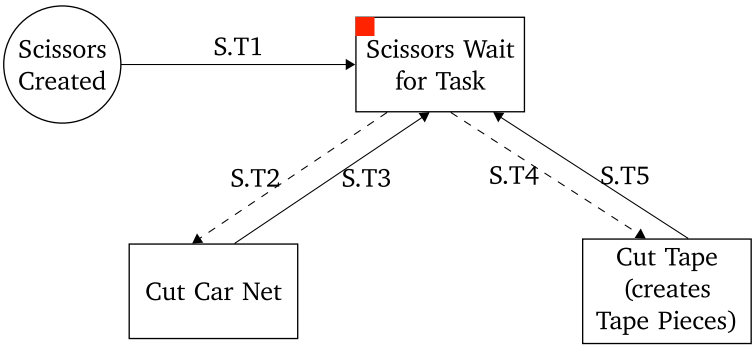

Figure 3.2: Scissors Activity Diagram

In the boxes below draw the activity diagrams for the remaining entities.

4 Radiology Clinic

4.1 Data

Table 4.1: List of Global Variables

Name	Description	Initial Value
NextPatIdNum	The Id number that will be assigned to the next patient	1
NextReceptionistIdNum	The Id number that will be assigned to the next receptionist	1
NextCTMachineIdNum	The Id number that will be assigned to the next CT Machine	1
P	The set of all patients	$\emptyset$
R	The set of all receptionists	$\emptyset$
C	The set of all CT Machines	$\emptyset$

Table 4.2: List of Data Modules

Name	Source	Identification	Input	Output
PatientInterarrivalTime	Poisson Process	Parameter	Mean interarrival time	Sample from Distribution
NumReceptionists	Constant	Parameter	N/A	Value
NumCTMachines	Constant	Parameter	N/A	Value
CheckInTime	Uniform Distribution	Parameter	Min and max time	Sample from Distribution
ScanTime	Log-normal Distribution	Parameter	Mean and std. dev.	Sample from Distribution

4.2 Components

Table 4.3: List of Entities

Entity	Attributes
Patient	ID State StateTimes
Receptionist	ID State StateTimes
CT Machine	ID State StateTimes

Table 4.4: List of Transitions

No.	Participant	From Event	To Event
1	Patient	Arrive(P)	Wait for check in.Start
2	Patient	Wait for check in.End	Check in.Start
3	Patient	Check in.End	Wait for scan.Start
4	Patient	Wait for scan.End	Scan.Start
5	Patient	Scan.End	Leave(P)
6	Receptionist	Arrive(R)	Wait for task(R).Start
7	Receptionist	Wait for task(R).End	Check in.Start
8	Receptionist	Check in.End	Wait for task(R).Start
9	Receptionist	Wait for task(R).End	Leave(R)
10	CT Machine	Arrive(CT)	Wait for task(CT).Start
11	CT Machine	Wait for task(CT).End	Scan.Start
12	CT Machine	Scan.End	Wait for task(CT).Start
13	CT Machine	Wait for task(CT).End	Leave(CT)

Table 4.5: Activities

Activity	Participants	Event	Type	State Change
Wait for Check In	Patient (p)	Start	Scheduled	`TRIGGER OnStartWaitForCheckIn WITH p`
		End	Controlled	
Check In	Patient (p), Receptionist (r)	Start	Controlled	`SCHEDULE Check In.End at TIME + CheckInTime()`
		End	Scheduled	`START Wait for Scan WITH p` `START Wait for Task (R) WITH r`
Wait for Scan	Patient (p)	Start	Scheduled	
		End	Controlled	`TRIGGER OnStartWaitForScan WITH p`
Scan	Patient (p), CTMachine (c)	Start	Controlled	`SCHEDULE Scan.End at TIME + ScanTime()`
		End	Scheduled	`START Leave (P) WITH p` `START Wait for Task (CT) WITH c`
Wait for Task (R)	Receptionist (r)	Start	Scheduled	`TRIGGER OnStartWaitForTaskR WITH r`
		End	Controlled	
Wait for Task (CT)	CTMachine (c)	Start	Scheduled	`TRIGGER OnStartWaitForTaskCT WITH c`
		End	Controlled	

4.3 Activity Diagrams

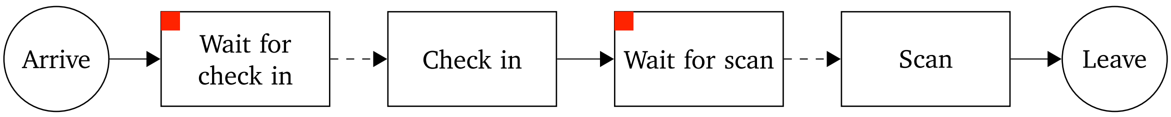

Figure 4.1: Patient Activity Diagram

Table 4.6: Events

Activity	Participants	Type	State Change

Simulation Start	-	Scheduled	`SCHEDULE Arrival (R) at TIME`
			`SCHEDULE Arrival (CT) at TIME`
			`SCHEDULE Arrival (P) at TIME + PatientInterArrival()`

Arrival (P)	Patient (p)	Scheduled	`p.ID = NextPatIDNum`
			`p.Priority = PatientPriority()`
			`NextPatIDNum = NextPatIDNum + 1`
			`SCHEDULE Arrival (P) at TIME + PatientInterArrival()`
			`START Wait for Check In WITH p`

Leave (P)	Patient (p)	Scheduled	`Calculate statistics for p`

Arrival (R)	Receptionist (r)	Scheduled	`r.ID = NextReceptionistIDNum`
			`NextReceptionistIDNum = NextReceptionistIDNum + 1`
			`IF NextReceptionistIDNum <= NumReceptionists THEN`
			`    SCHEDULE Arrival (R) at TIME`
			`END IF`
			`START Wait for Task (R) WITH r`

Leave (R)	Receptionist (r)	Scheduled	`Calculate statistics for r`

Arrival (CT)	CTMachine (c)	Scheduled	`c.ID = NextCTMachineIDNum`
			`NextCTMachineIDNum = NextCTMachineIDNum + 1`
			`IF NextCTMachineIDNum <= NumCTMachines THEN`
			`    SCHEDULE Arrival (CT) at TIME`
			`END IF`
			`START Wait for Task (CT) WITH c`

Leave (P)	CTMachine (c)	Scheduled	`Calculate statistics for c`

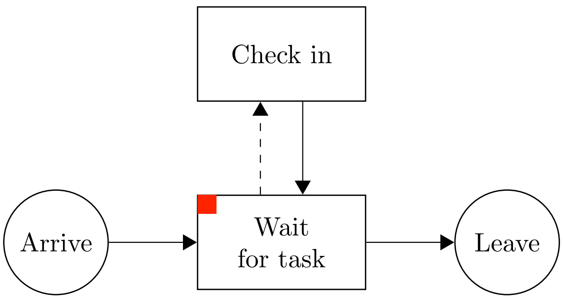

Figure 4.2: Receptionist Activity Diagram

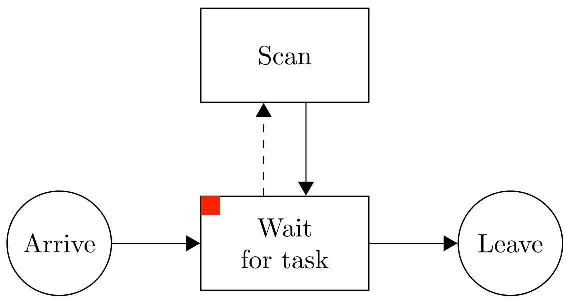

Figure 4.3: CT Activity Diagram

4.4 Control Policies

Table 4.7: OnStartWaitForCheckIn

Triggered by: Patient p

```
receps = {r FOR r IN R IF r.State = "Wait for task (R)"}
IF receps IS NOT empty THEN
    r_hat = argmin{r.CurrentStart FOR r IN receps}
    START Check In WITH p, r_hat
END IF
```

Table 4.8: OnStartWaitForScan

Triggered by: Patient p

```
cts = {c FOR c IN C IF c.State = "Wait for task (C)"}
IF cts IS NOT empty THEN
    c_hat = argmin{c.CurrentStart FOR c IN cts}
    START Scan WITH p, r_hat
END IF
```

Table 4.9: OnStartWaitForTaskR

Triggered by: Receptionist r

```
patients = {p FOR p IN P IF p.State = "Wait for Check In"}
IF patients IS NOT empty THEN
    p_hat = argmin{p.CurrentStart FOR p IN patients}
    START Check In WITH p, r_hat
END IF
```

Table 4.10: OnStartWaitForTaskCT

Triggered by: CTMachine c

```
patients = {p FOR p IN P IF p.State = "Wait for Scan"}
IF patients IS NOT empty THEN
    p_hat = argmin{p.CurrentStart FOR p IN patients}
    START Scan WITH p, r_hat
END IF
```