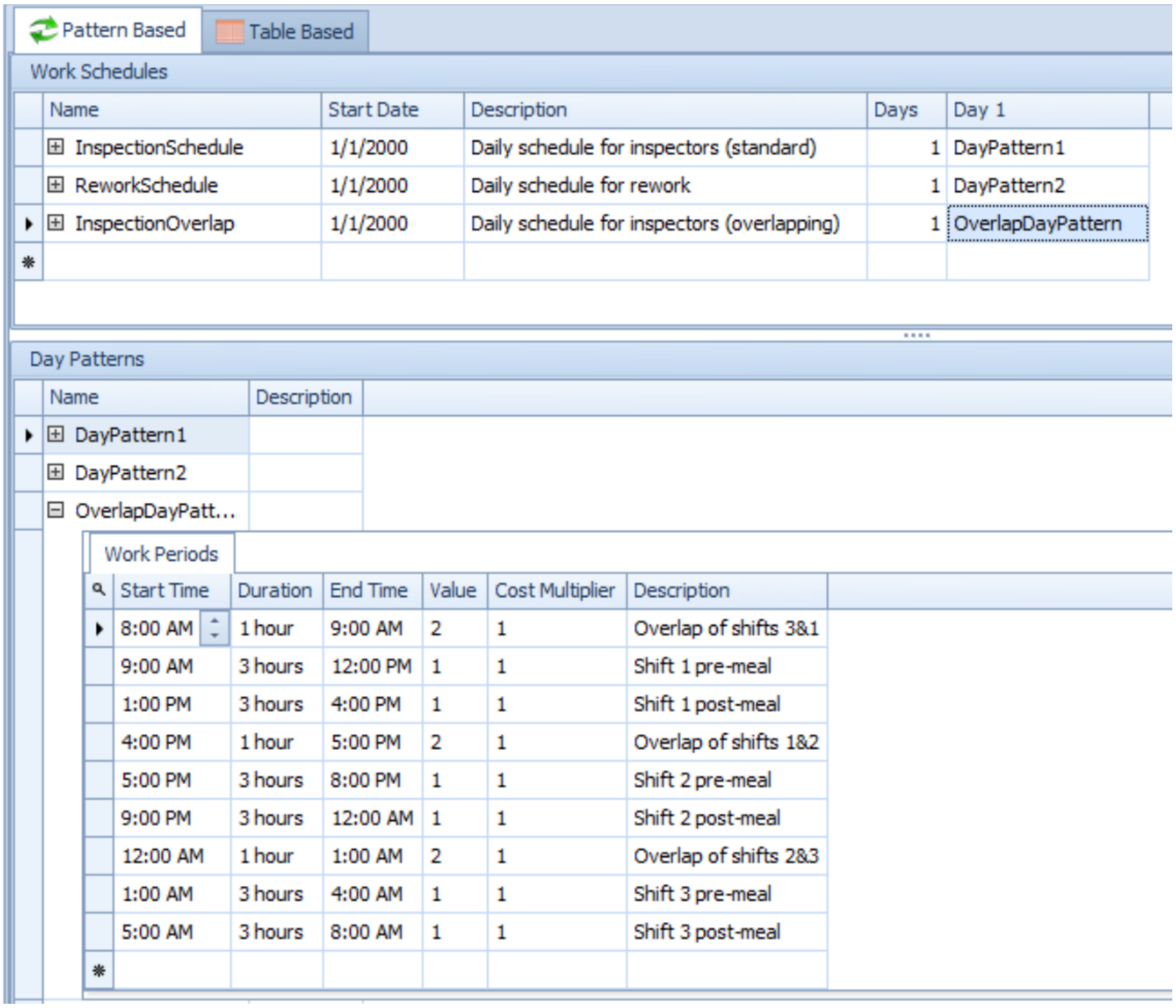
DATA 604 Assignment 7: Intermediate Modeling 2

Dan Smilowitz

March 17, 2017

Problem 6

Model 5.2 is established using the fixed weights of 66%, 8%, and 26% for good, bad, and reworked parts, respectively. To allow for investigation of the overlap of the three shifts, a new schedule (*InspectionOverlap*) is created using a new day pattern (*OverlapDayPattern*):



Pattern Based | Table Based

Work Schedules

Name	Start Date	Description	Days	Day 1
InspectionSchedule	1/1/2000	Daily schedule for inspectors (standard)	1	DayPattern1
ReworkSchedule	1/1/2000	Daily schedule for rework	1	DayPattern2
InspectionOverlap	1/1/2000	Daily schedule for inspectors (overlapping)	1	OverlapDayPattern

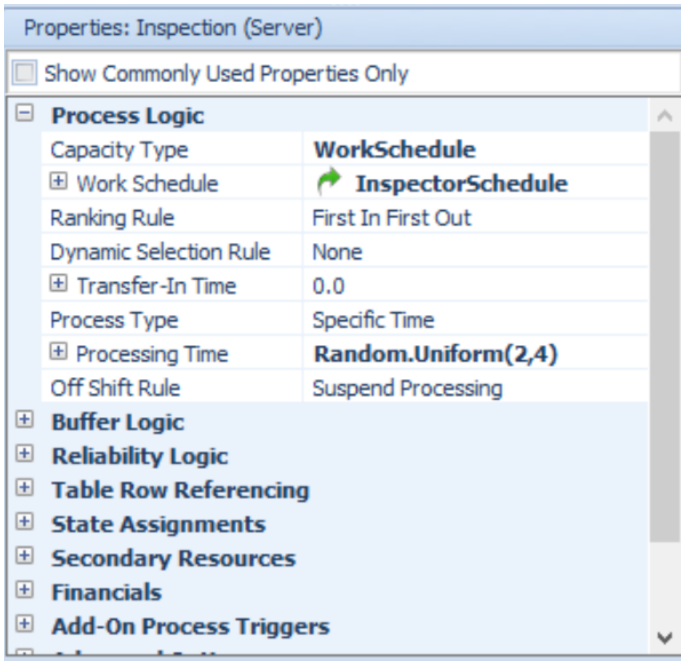
Day Patterns

Name	Description
DayPattern1	
DayPattern2	
OverlapDayPatt...	

Work Periods

Start Time	Duration	End Time	Value	Cost Multiplier	Description
8:00 AM	1 hour	9:00 AM	2	1	Overlap of shifts 3&1
9:00 AM	3 hours	12:00 PM	1	1	Shift 1 pre-meal
1:00 PM	3 hours	4:00 PM	1	1	Shift 1 post-meal
4:00 PM	1 hour	5:00 PM	2	1	Overlap of shifts 1&2
5:00 PM	3 hours	8:00 PM	1	1	Shift 2 pre-meal
9:00 PM	3 hours	12:00 AM	1	1	Shift 2 post-meal
12:00 AM	1 hour	1:00 AM	2	1	Overlap of shifts 2&3
1:00 AM	3 hours	4:00 AM	1	1	Shift 3 pre-meal
5:00 AM	3 hours	8:00 AM	1	1	Shift 3 post-meal

With two schedules now set up for inspection workers, the work schedule for the Inspection server is replaced by a reference to the newly-created reference property *InspectorSchedule*:



Properties: Inspection (Server)

☐ Show Commonly Used Properties Only

⊟ **Process Logic**

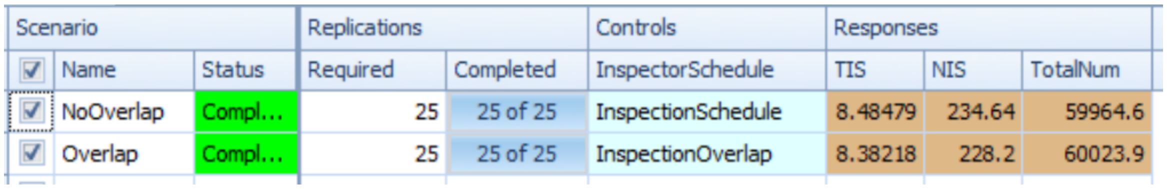
Capacity Type	**WorkSchedule**
⊞ Work Schedule	**InspectorSchedule**
Ranking Rule	First In First Out
Dynamic Selection Rule	None
⊞ Transfer-In Time	0.0
Process Type	Specific Time
⊞ Processing Time	**Random.Uniform(2,4)**
Off Shift Rule	Suspend Processing

⊞ **Buffer Logic**
⊞ **Reliability Logic**
⊞ **Table Row Referencing**
⊞ **State Assignments**
⊞ **Secondary Resources**
⊞ **Financials**
⊞ **Add-On Process Triggers**

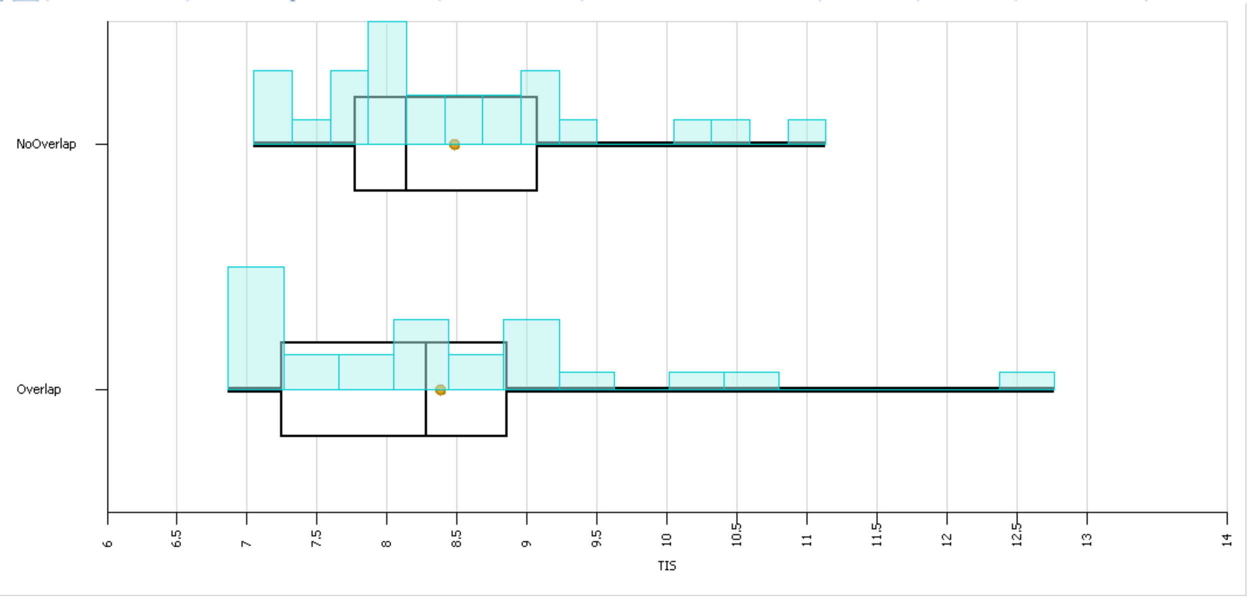
An experiment is then run with two scenarios (one for each schedule) for 25 replications each of 300 days with a 50-day warmup period. Three responses are created and tracked:

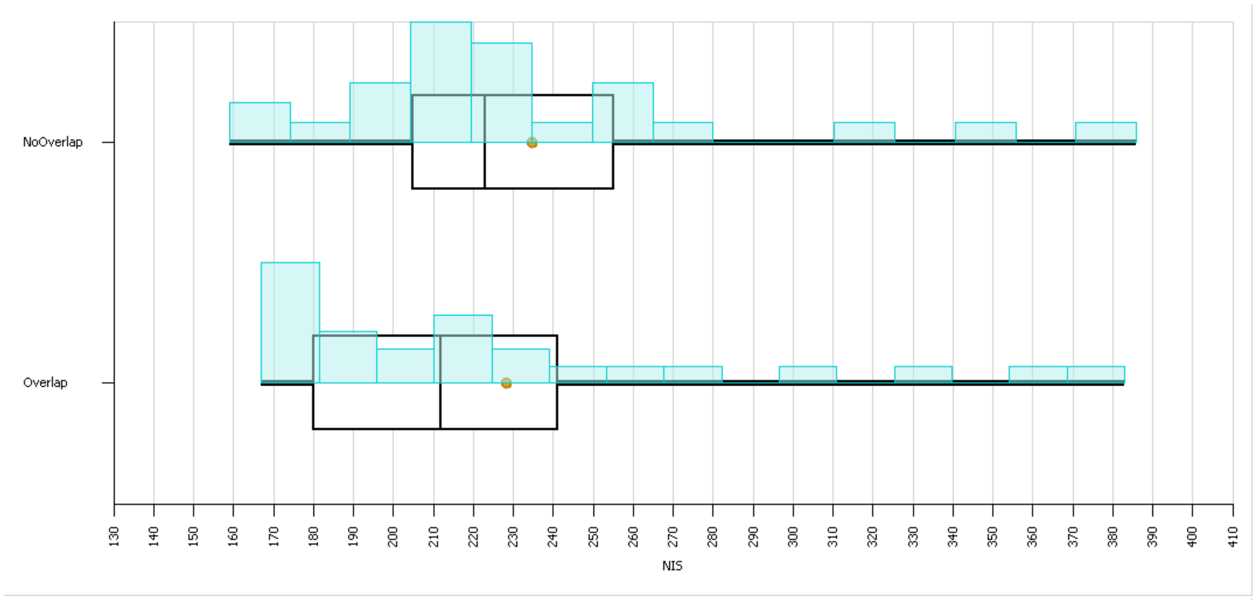
- TIS: `PCB.Population.TimeInSystem.Average`
- NIS: `PCB.Population.NumberInSystem.Maximum`
- TotalNum: `PCB.Population.NumberDestroyed`

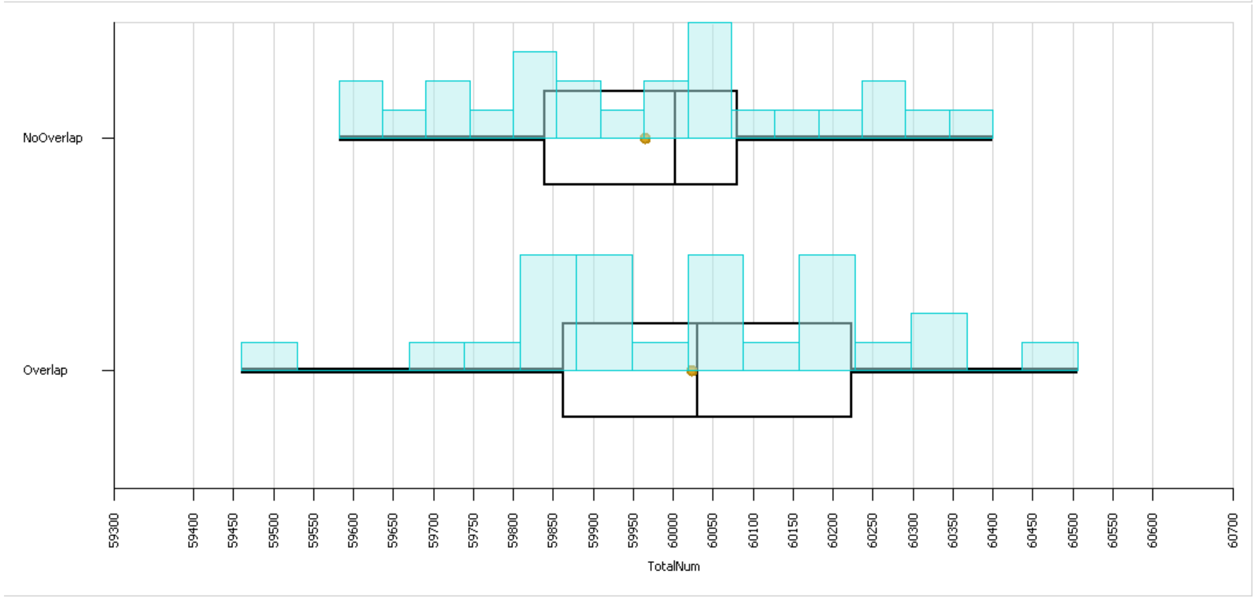
The objective of these responses are set to be minimum, minimum, and maximum, respectively. The results of the experiment, along with SMORE plots, are presented below:



Scenario		Replications		Controls	Responses		
Name	Status	Required	Completed	InspectorSchedule	TIS	NIS	TotalNum
✓ NoOverlap	Compl...	25	25 of 25	InspectionSchedule	8.48479	234.64	59964.6
✓ Overlap	Compl...	25	25 of 25	InspectionOverlap	8.38218	228.2	60023.9





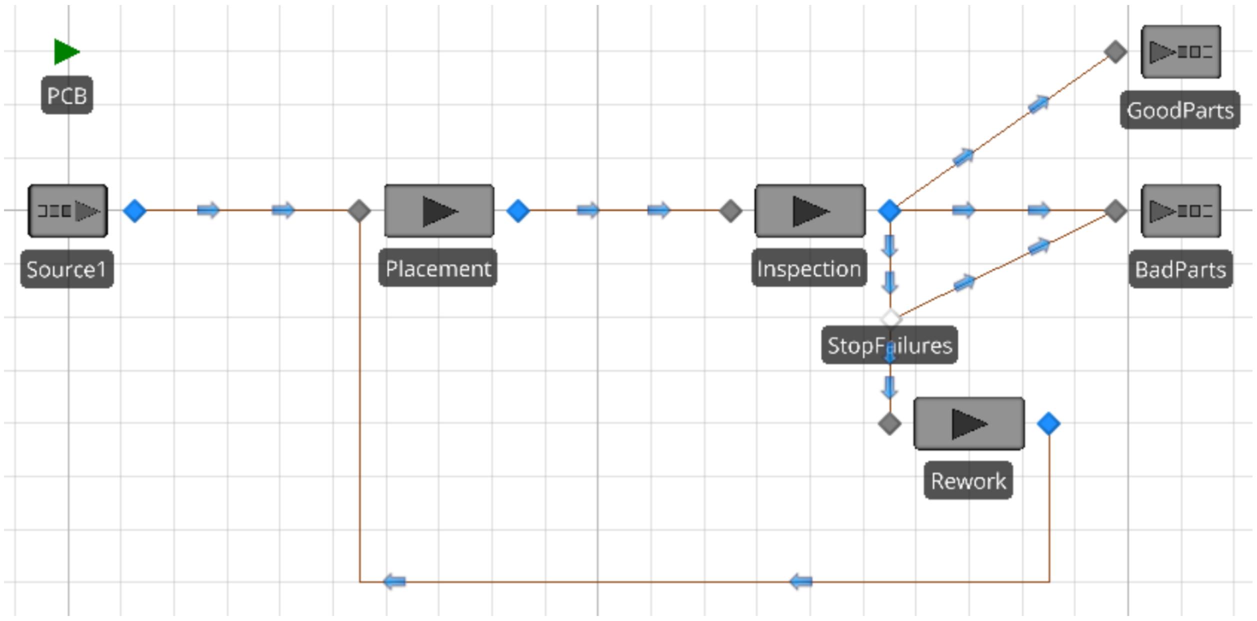


This experiment shows that having the schedules of inspectors does not improves performance of the model. This makes intuitive sense since the new schedule yields 24 total hours of inspector availability, as opposed to 21 total hours without overlap.

Problem 7

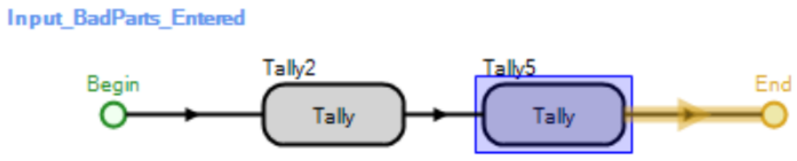
In order to declare parts failing inspection 3 times bad parts, a new discrete entity state, *RepeatFailure*, is created. A tally statistic, *NumRepeatFailures*, is created in the model to track the number of repeat failures.

To ensure that parts processed 3 times that would have otherwise gone for rework (i.e. failed inspection 3 times) are instead classified as bad parts, a node is added between the Inspection server and the Rework server. The weight of the connection to this node is 0.26, continuing to reflect that 26% of parts will need reworking. The outputs of this node are connected to the Rework server and BadParts sink:

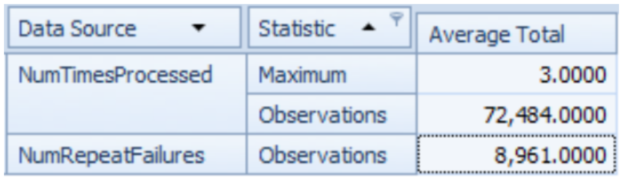


The weights of the connectors to the rework and bad parts are `ModelEntity.TimesProcessed < 3` and `ModelEntity.TimesProcessed == 3`, respectively, and the outbound link rule of the node is set by link weight. Thus, when a part is sent for rework, it will be reworked if it has been inspected fewer than 3 times, but will automatically be deemed a bad part if it has been processed 3 times. If the part has been inspected 3 times, the node sets *RepeatFailure* to 1 (true) prior to sending it to the sink.

The add-on process for the input node of the BadParts sink is modified to tally the number of parts deemed bad due to exceeding the number of acceptable inspections:



The results of running this model for 300 days are presented below:



Data Source	Statistic	Average Total
NumTimesProcessed	Maximum	3.0000
	Observations	72,484.0000
NumRepeatFailures	Observations	8,961.0000

The first row of the pivot grid shows that the model succesfully limited the number of times a part could be inspected to 3. 8,961 of the 72,484 parts processed (12.4%) were rejected due to 3 inspection failures.

Problem 8

As shown for Model 5-2 in the text, the output rate of the placement machine is equal to the incoming rate since $\rho_{placement} < 1$. This outgoing rate is the incoming rate for the fine-pitch machines:

$$\lambda_{fine} = \lambda' = 1.3514\lambda = 13.514$$

The expected service times of the three fine-pitch machines can be used to calculate the expected utilzation of the three stations:

$$\begin{aligned} E(S_{fast}) = 9 &\rightarrow \mu_{fast} = 6.6667 \rightarrow \rho_{fast} = 2.0271 \\ E(S_{med}) = 12 &\rightarrow \mu_{fast} = 5.0000 \rightarrow \rho_{med} = 2.7028 \\ E(S_{slow}) = 14 &\rightarrow \mu_{fast} = 4.2857 \rightarrow \rho_{fast} = 3.1533 \end{aligned}$$

These utilizations are all greater than one — all of the machines will have more work than they can complete! This is why selection of a fine-pitch machine is based on *overload* – the amount by which the machines' capacities are exceeded.

The fast fine-pitch machine has is subjecto to failures every 3 hours, which take 30 minutes to repair. This needs to be accounted for, since the utilization of the machine is strictly greater than 1. Since the machine is only available for 180 of every 210 minutes, the new effective utilization for the fast machine becomes

$$\rho'_{fast} = \rho_{fast} \times \frac{210}{180} = 2.3650$$

Since selection of machines is based on overload at each machine when a part exits the placement machine, a measure of the overload must be generated. It is reasonable to assume that overload is inversely proportional to utilization:

$$OL_x \propto \frac{1}{\rho_x} \quad \forall \quad x \in \{fast, med, slow\}$$

To get the proportion of parts going to each of the machines, the ratio of the individual machine overloads to the total overloads is taken:

$$p_x = \frac{OL_x}{\sum OL} \quad \forall \quad x \in \{fast, med, slow\}$$

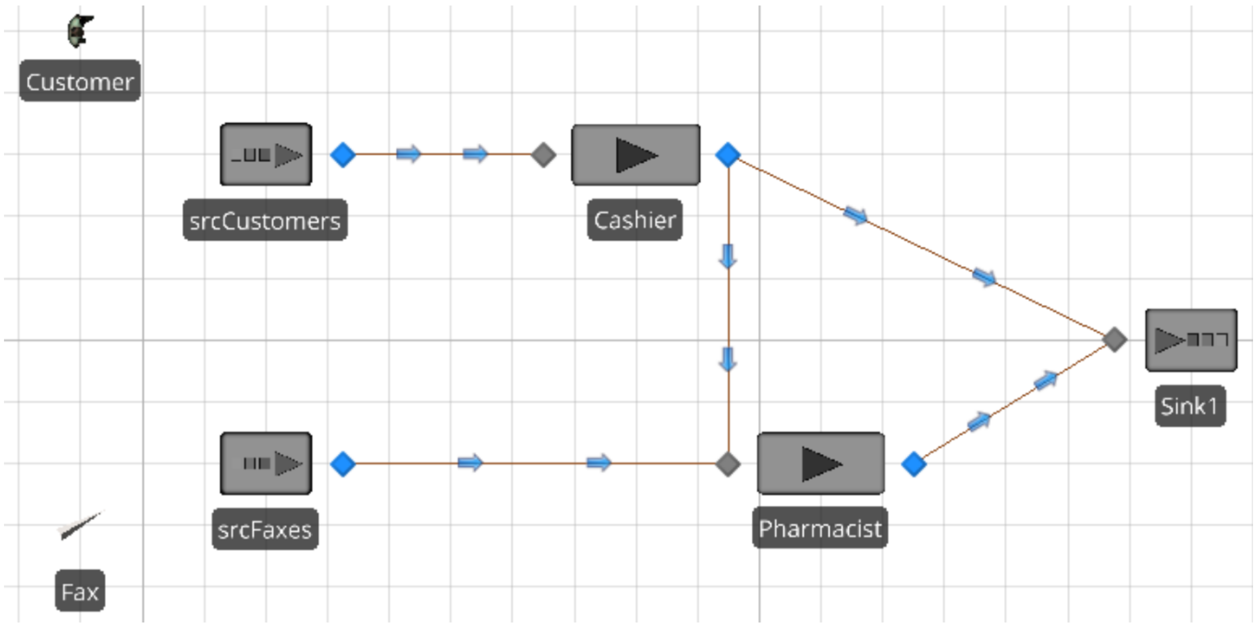
Substituting in the calculated values of ρ into the prior two equations yields the following results:

$$\begin{aligned} p_{fast} &= 0.3809 \\ p_{med} &= 0.3333 \\ p_{slow} &= 0.2857 \end{aligned}$$

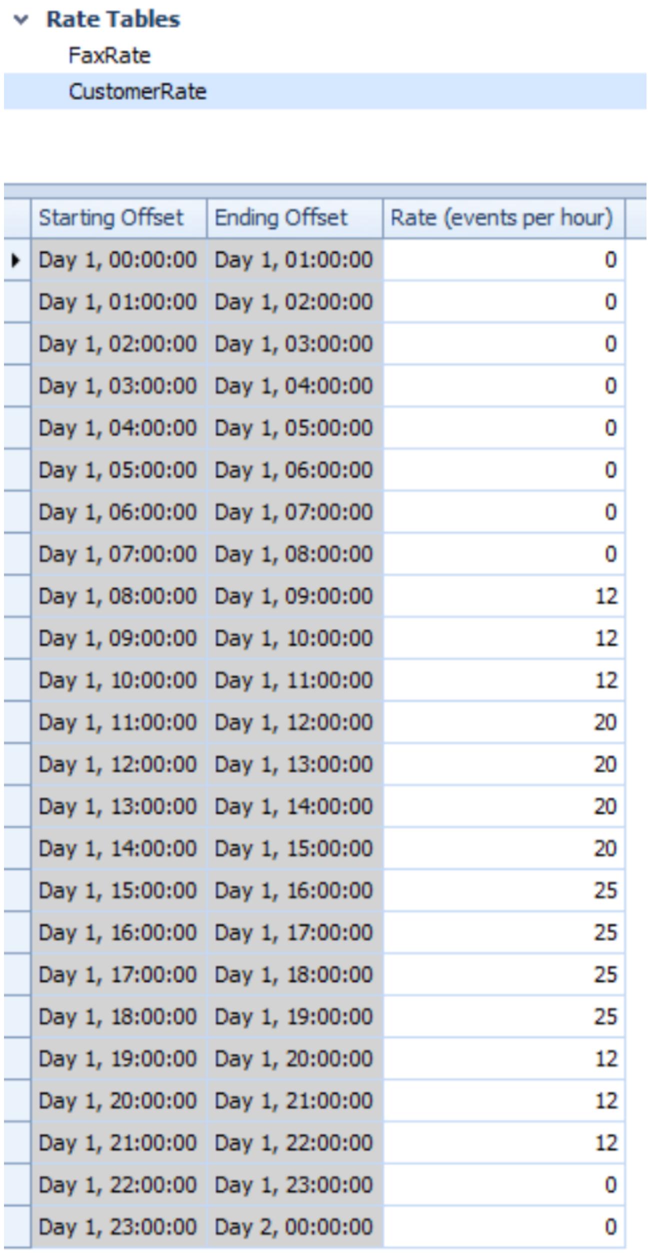
These match the stated proprtions of 38%, 33%, and 29%.

Problem 9

In order to model the described pharmacy, the following model is designed:

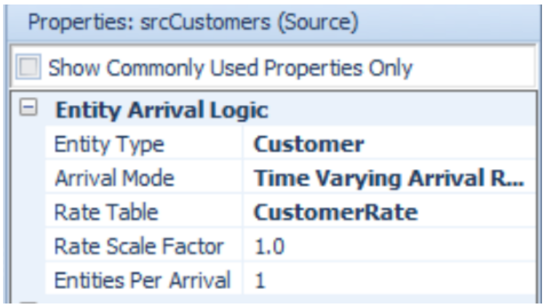


The model has two entities, *Customers* and *Prescriptions* – these represent customers arriving and prescriptions faxed in, respectively. Each of the entities has a source object associated with it (*srcCustomers* and *srcFaxes*). These source objects are each assigned the appropriate entity type. Rate tables are assigned for each of the types, and the arrival mode is pointed to the appropriate rate table. These are shown for the *srcCustomers* source below:



Rate Tables
FaxRate
CustomerRate

Starting Offset	Ending Offset	Rate (events per hour)
Day 1, 00:00:00	Day 1, 01:00:00	0
Day 1, 01:00:00	Day 1, 02:00:00	0
Day 1, 02:00:00	Day 1, 03:00:00	0
Day 1, 03:00:00	Day 1, 04:00:00	0
Day 1, 04:00:00	Day 1, 05:00:00	0
Day 1, 05:00:00	Day 1, 06:00:00	0
Day 1, 06:00:00	Day 1, 07:00:00	0
Day 1, 07:00:00	Day 1, 08:00:00	0
Day 1, 08:00:00	Day 1, 09:00:00	12
Day 1, 09:00:00	Day 1, 10:00:00	12
Day 1, 10:00:00	Day 1, 11:00:00	12
Day 1, 11:00:00	Day 1, 12:00:00	20
Day 1, 12:00:00	Day 1, 13:00:00	20
Day 1, 13:00:00	Day 1, 14:00:00	20
Day 1, 14:00:00	Day 1, 15:00:00	20
Day 1, 15:00:00	Day 1, 16:00:00	25
Day 1, 16:00:00	Day 1, 17:00:00	25
Day 1, 17:00:00	Day 1, 18:00:00	25
Day 1, 18:00:00	Day 1, 19:00:00	25
Day 1, 19:00:00	Day 1, 20:00:00	12
Day 1, 20:00:00	Day 1, 21:00:00	12
Day 1, 21:00:00	Day 1, 22:00:00	12
Day 1, 22:00:00	Day 1, 23:00:00	0
Day 1, 23:00:00	Day 2, 00:00:00	0



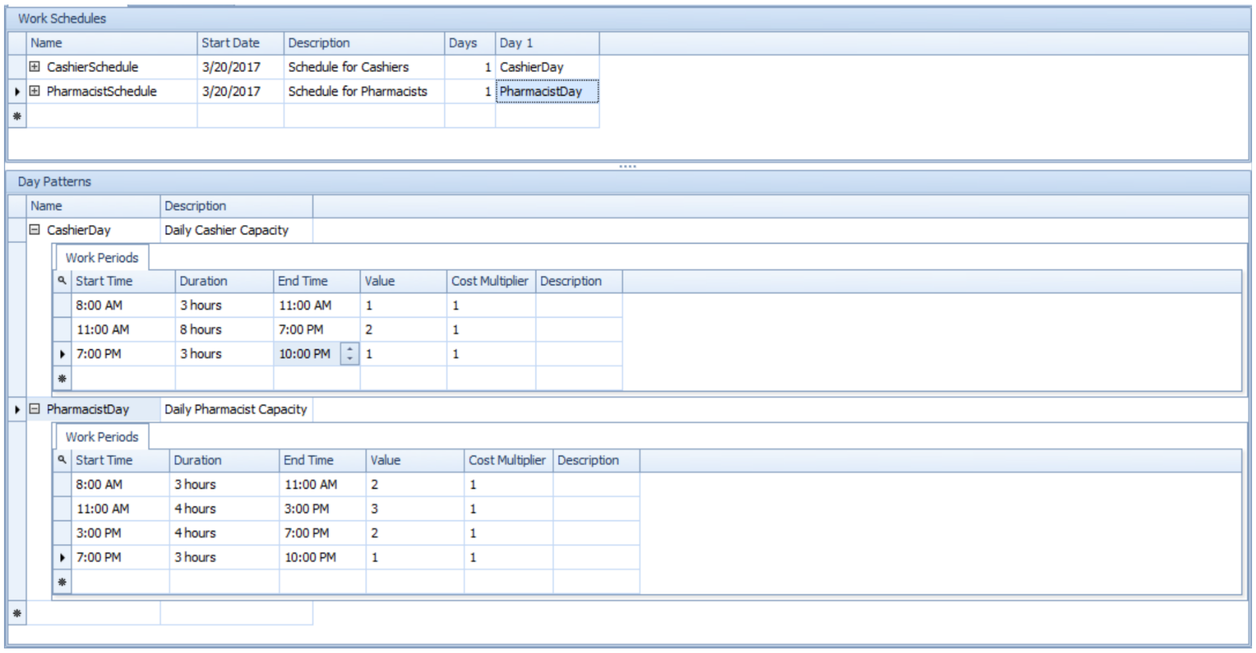
Properties: srcCustomers (Source)

Show Commonly Used Properties Only

Entity Arrival Logic

Entity Type	**Customer**
Arrival Mode	**Time Varying Arrival R...**
Rate Table	**CustomerRate**
Rate Scale Factor	1.0
Entities Per Arrival	1

The capacity of the two servers (*Cashier* and *Pharmacist*) are determined based on the given calendar-based schedules, as illustrated below:



Work Schedules

Name	Start Date	Description	Days	Day 1
CashierSchedule	3/20/2017	Schedule for Cashiers	1	CashierDay
PharmacistSchedule	3/20/2017	Schedule for Pharmacists	1	PharmacistDay

Day Patterns

Name	Description
CashierDay	Daily Cashier Capacity

Work Periods

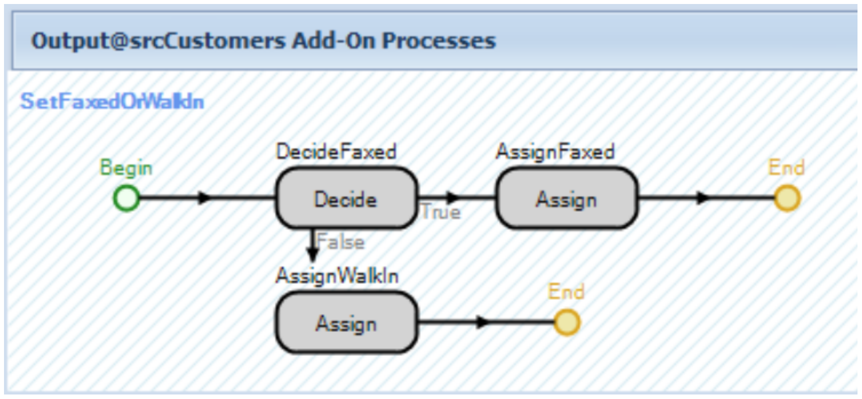
Start Time	Duration	End Time	Value	Cost Multiplier	Description
8:00 AM	3 hours	11:00 AM	1	1	
11:00 AM	8 hours	7:00 PM	2	1	
7:00 PM	3 hours	10:00 PM	1	1	

Name	Description
PharmacistDay	Daily Pharmacist Capacity

Work Periods

Start Time	Duration	End Time	Value	Cost Multiplier	Description
8:00 AM	3 hours	11:00 AM	2	1	
11:00 AM	4 hours	3:00 PM	3	1	
3:00 PM	4 hours	7:00 PM	2	1	
7:00 PM	3 hours	10:00 PM	1	1	

A new boolean entity state variable *IsFaxed* is defined – this variable tracks if an arriving customer entity has faxed their prescription in ahead of time. This variable is assigned using an add-on process for the exit node of the *Customers* entity – this randomly assigns 59% of customers a value of True and 41% a value of False.



This variable is used to determine the path of customers. All customers first go to the cashier, the output node of the cashier has two connectors: one leading to the pharmacist for walk-in customers, and one leading to the sink for fax-ahead customers. This logic is implemented by setting the selection weight of both connectors to `ModelEntity.IsFaxed`; this means those faxed simply leave, while those not faxed must go to the pharmacist.

This model makes some simplifying assumptions:

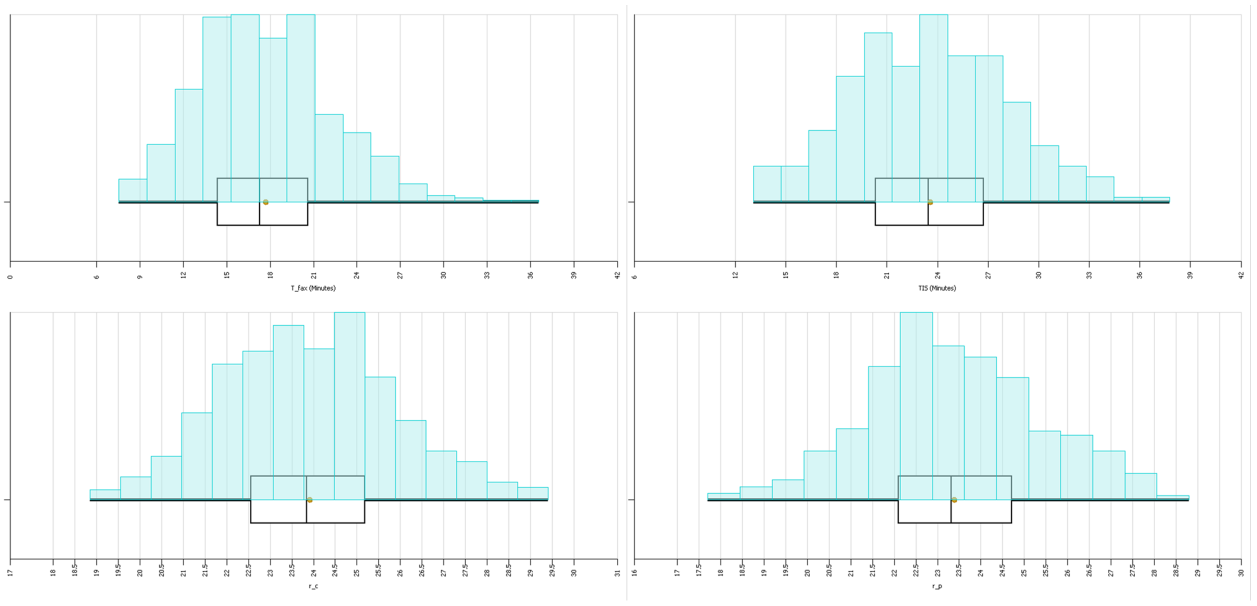
- Each customer (and therefore each fax) is only filling one prescription at a time
- There is no time for transportation of faxed-in prescriptions to the cashier

A more advanced model may incorporate varying numbers of prescriptions, as well as adding a step to match entities of multiple types (prescriptions and customers).

An experiment of 500 replications, each running from 8:00 AM to 10:00 PM, is established with the following measures:

- T__fax = `Fax.Population.TimeInSystem.Average`
- TIS = `Customer.Population.TimeInSystem.Average`
- r__c = `Cashier.Capacity.ScheduledUtilization`
- r__p = `Pharmacist.Capacity.ScheduledUtilization`

The results of this experiment are visualized below:



The mean time for faced prescriptions to be filled by the pharmacist is 17.71 minutes; the average total time in the system for customers is 23.57. The utilization of the cashier and pharmacist are 23.91% and 23.39%, respectively.