Computer Simulation

CS 312L

LAB MANUAL

Department of Industrial Engineering

School of Engineering

University of Management and Technology, Lahore

Prepared by: (Name)	Hafiz Osaid Date
Supervise by: (Name)	
Checked by: (Name)	
Approved by: (Name)	

CS 312L Course Learning Outcomes (CLOs)

On the completion of this course, student will be able to:

1. Understand the fundamental logic, structure, components of simulation and apply the basic simulation concepts to model the activities of a given system in ARENA. (C**3*,CLO 1**)
2. Pursue effectively with Arena to build simple or complex model on his own.(**A 3*, CLO2**)
3. Develop a spreadsheet model to illustrate some of the key elements of any simulation model. (**C6*,CLO3**)

Note: *P, A, C stand for psychomotor, affective, cognitive domains of Bloom's taxonomy respectively. The numeric value refers to the corresponding level in each domain.For details regarding Bloom's taxonomy, please see the annexure at the end of the manual.

Mapping of CLOs to Program Learning Outcomes (PLOs):

Semester	Course Code	Title	Course Learning Outcomes	PLO1 Engg. Knowledge	PLO2 Problem Analysis	PLO3 Solution Design	PLO4 Investigation	PLO5 Mod. Tool Usage	PLO6 Engr. & Society	PLO7 Env. & Sust.	PLO8 Ethics	PLO9 Team Work	PLO10 Communication	PLO11 Proj. Mgmt.	PLO12 Lifelong Learning
Fifth	CS312L	Computer Simulations (Lab)	Understand the fundamental logic, structure, components of simulation and apply the basic simulation concepts to model the activities of a given system in ARENA	✔											
			Pursue effectively with Arena to build simple or complex model on his own.			✔									
			Develop a spreadsheet model to illustrate some of the key elements of any simulation model.		✔										

Grade Evaluation Criteria

Components	Marks
Lab Sessional	50%
Final Performance	25%
Final Viva	25%
Total	100%

Exp. No	EXPERIMENTS	CLO
1	Introduction to computer simulation and ARENA software.	1
2	To understand the fundamental Simulation concepts by using input analyzer.	1
3	To build simple process system model by using ARENA software.	2
4	To build generalizes series and parallel loan application model by using ARENA**.**	2
5	To develop monte carlo simulation for a drilling center.	2
6	To Simulate and analyze a bank problem.	2
7	To Simulate and analyze an airport security system.	2
8	Simulate and analyze a small manufacturing system with two processes in parallel.	2
9	To build simulation model of a hospital emergency room.	2
10	Construct simulation model for some food restaurant.	2
11	To provide plan for car maintenance through simulation.	2

12	To simulate a coin toss in spreadsheet.	3
13	To Simulate a Random service time by using spreadsheet.	3

Lab Session No.1

Introduction to computer simulation and ARENA software.

Theory

Simulation

Simulation refers to a broad collection of methods and applications to mimic the behavior of real systems, usually on a computer with appropriate software. In fact, "simulation" can be an extremely general term since the idea applies across many fields, industries, and applications. These days, simulation is more popular and powerful than e

ver since computers and software are better than ever.

Computer Simulation

Computer simulation refers to methods for studying a wide variety of models of realworld systems by numerical evaluation using software designed to imitate the system's operations or characteristics, often over time. From a practical viewpoint, simulation is the process of designing and creating a computerized model of a real or proposed system for the purpose of conducting numerical experiments to give us a better understanding of the behavior of that system for a given set of conditions. Although it can be used to study simple systems, the real power of this technique is fully realized when we use it to study complex systems. While simulation may not be the only tool you could use to study the model, it's frequently the method of choice. The reason for this is that the simulation model can be allowed to become quite complex, if needed to represent the system faithfully, and you can still do a simulation analysis. Other methods may require stronger simplifying assumptions about the system to enable an analysis, which might bring the validity of the model into question.

Different Kinds of Simulation Models

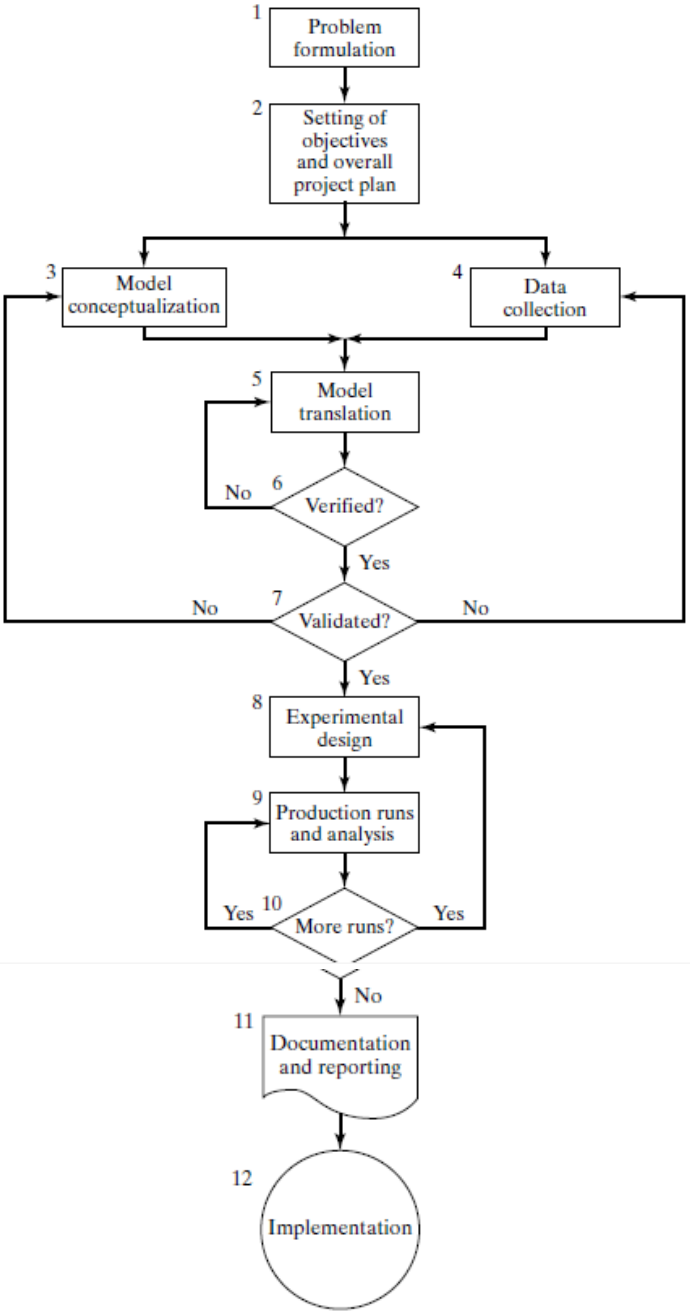
- Static
- Dynamic
- Continuous vs. Discret

- **Deterministic vs. Stochastic:**

Models that have no random input are deterministic; a strict appointment-book operation with fixed service times is an example. Stochastic models, on the other hand, operate with at least some inputs being random—like a bank with randomly arriving customers requiring varying service times.

Steps in a Simulation Study

Figure 3 shows a set of steps to guide a model builder in a thorough and sound simulation study.Similar figures and discussion of steps can be found in other sources [Shannon, 1975; Gordon, 1978; Law, 2007].The steps in a simulation study are shown in .

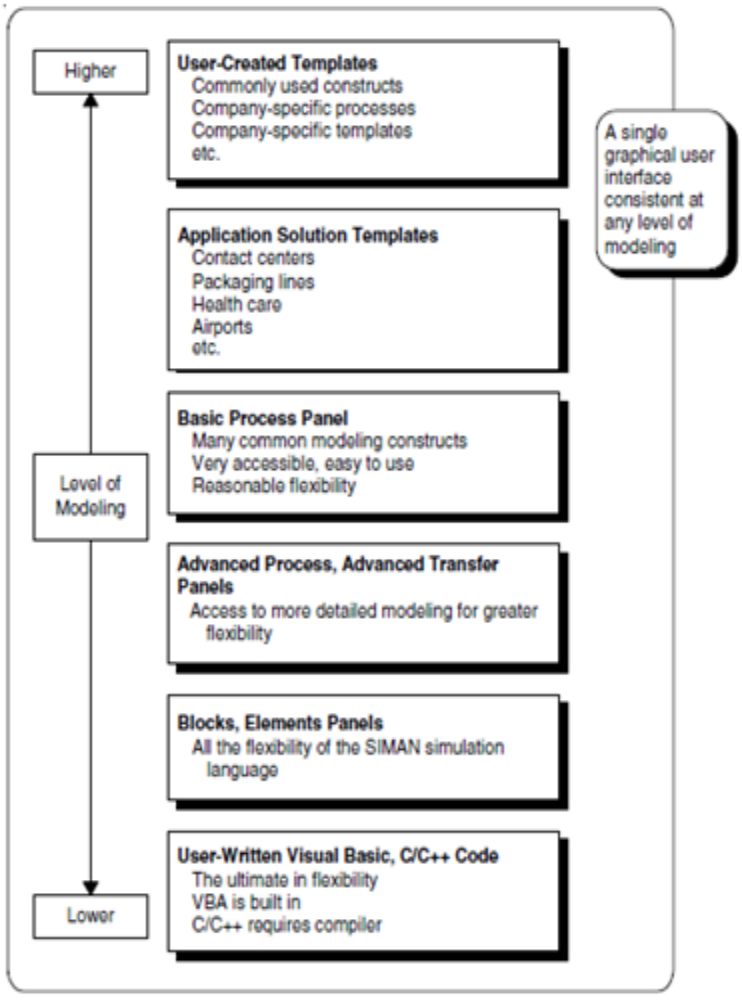


Steps in a simulation study.

Where Arena Fits In

Figure SEQ Figure * ARABIC 1: Simulation study steps

Arena combines the ease of use found in high-level simulators with the flexibility of simulation languages and even all the way down to general-purpose procedural languages like the Microsoft® Visual Basic® programming system or C if you really want. It does this by providing alternative and interchangeable *templates* of graphical simulation modeling and analysis *modules* that you can combine to build a fairly wide variety of simulation models. For ease of display and organization, modules are typically grouped into *panels* to compose a template. By switching panels, you gain access to a whole different set of simulation modeling constructs and capabilities. In most cases, modules from different panels can be mixed together in the same model.



Arena's Hierarchical Structure

Figure SEQ Figure * ARABIC 2: Arena hierarchical structure

Arena maintains its modeling flexibility by being fully *hierarchical*, as depicted in Figure 1-2. At any time, you can pull in low-level modules from the Blocks and Elements panel and gain access to simulation-language flexibility if you need to and mix in SIMAN constructs together

with the higher-level modules from other templates (Arena is based on, and actually includes, the SIMAN simulation language; see Pedgen, Shannon, and Sadowski 1995 for a complete discussion of SIMAN). For specialized needs, like complex decision algorithms or accessing data from an external application, you can write pieces of your model in a procedural language like Visual Basic or C/C++. All of this, regardless of how high or low you want to go in the hierarchy, takes place in the same consistent graphical user interface.

Learning "ARENA"

Arena Basic Edition software lets you bring the power of modeling and simulation to business process improvement. It is designed primarily for newcomers to simulation and serves as an introductory product and foundation to the rest of the Arena product family.
Typically, any process that can be described by means of a flowchart can be simulated with Arena Basic Edition.
Arena Basic Edition is most effective when analyzing business, service, or simple (nonmaterial-handling intensive) manufacturing processes or flows.
Typical scenarios include:

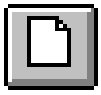
- Documenting, visualizing, and demonstrating the dynamics of a process with animation
- Predicting system performance based on key metrics such as costs, throughput, cycle times, and utilizations
- Identifying process bottlenecks such as queue build ups and over-utilization of resources
- Planning staff, equipment, or material requirements

In addition to the Arena Basic Edition, Rockwell Automation offers a full suite of products to provide enterprise-wide simulation, optimization, and 3D model animation

- **Standard Tool Bar:**

In knowing the user interface of ARENA, the first step is to get the knowledge of different commands and their signs. The first in this respect is "STANDARD" tool bar. The different commands used in standard bar are described next.

- **NEW Command:**



Use this command to create a new document in Arena.
CTRL+N

- **OPEN Command:**

Use this command to open an existing document. You can have multiple files open concurrently; use the Window menu, 1, 2, 3, 4, commands to switch among the multiple open documents.
Keys: CTRL+O

- **TEMPLATE PANEL ATTACH COMMAND:**

The Template Panel Attach command will allow you to attach a model-building panel to the Project bar. A standard dialog will appear, allowing you to specify the name and location of the file you want to attach.

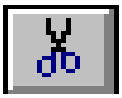
- **TEMPLATE_PANEL DETACH COMMAND:**

The Template Panel Detach command will allow you to detach a model-building panel from the Project bar. The template panel that is currently active will be detached.

- **PRINT COMMAND:**

Use this command to print contents of the model workspace. Only the contents of the active hierarchy level are considered.
Keys: CTRL+P

- **CUT COMMAND**



The Cut command removes the selected item or group of items from the window and places them in the clipboard.
Keys: CTRL+X

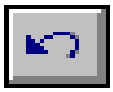
- **COPY COMMAND:**

The Copy command copies the selection to the clipboard.
Keys: CTRL+C

- **PASTE COMMAND:**

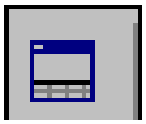
The Paste command pastes the contents of the clipboard into the window.
Keys: CTRL+V

- **UNDO COMMAND:**



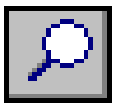
The Undo command allows you to undo previous actions.
Keys: CTRL+Z

- **SPLIT SCREEN COMMAND:**



Use this option to toggle the Arena window between full view and split-screen view. Split screen view displays both the model workspace (on top) and the spreadsheet (on bottom) in the window at once.

- **VIEW REGION COMMAND:**



Use this command to zoom in to a particular region of the window. When you select *View, Region*, use the cross-hair cursor to draw a box outlining the region you want to display.

- **LAYERS COMMAND**

The *Layers* option is used to toggle the display of Arena objects. If checked, objects of that type are displayed; if not checked, the objects are not displayed, though they still exist in the window.

ADD SUBMODEL COMMAND

Use this command to place an empty sub model into the current model view. Once placed, the name of the sub model will be appearing on the Project bar's Navigate panel.

- **CONNECT COMMAND**

The Connect command is used to connect two objects together. When this command is executed, the mouse cursor is changed to a cross hair with a "Connect" icon.
To connect two modules or submodel objects together, you first move the cursor to the exit point of the first object to connect. A valid exit point will be highlighted when the mouse cursor is hovered over it. Click on the exit point. Then move the cursor to the entry point of the second object to connect. A valid entry point will be highlighted when

the mouse cursor is hovered over it. Click on the entry point to create a connection between the two objects.

Comments

Lab Session No.2

To understand the fundamental Simulation concepts by using input analyzer.

Theory

Fundamental Simulation concepts

we introduce some of the underlying ideas, methods, and issues in simulation before getting into the Arena software itself These concepts are the same across any kind of simulation software, and some familiarity with them is essential to understanding how Arena simulates a model you've built.

Example

In this session, we describe the example system and decide what we'd like to know about its behavior and performance.

The System

Since a lot of simulation models involve waiting lines or *queues* as building blocks, we'llstart with a very simple case of such a model representing a portion of a manufacturing facility. "Blank" parts arrive to a drilling center, are processed by a single drill press, and then leave; see Figure 3. If a part arrives and finds the drill press idle, its processing at the drill press starts right away; otherwise, it waits in a First-In First-Out (FIFO) queue. This is the *logical* structure of the model.

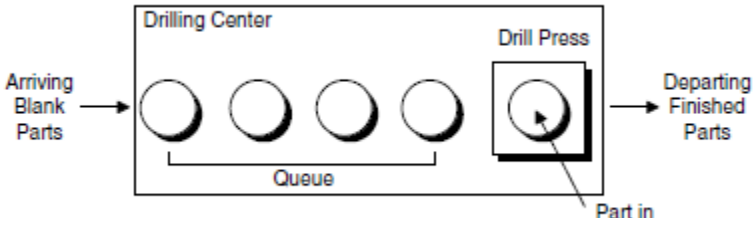


Figure SEQ Figure * ARABIC 3:A simple processing system

You have to specify the *numerical* aspects as well, including how the simulation starts and stops. First, decide on the underlying "base" units with which time will be measured; we'll use minutes for all time measurements here. It doesn't logically matter what the time units are, so pick whatever is most appropriate, familiar, and convenient for your application.1 You can express input time quantities in different units if it's customary or convenient, like minutes for mean service times but hours for mean machineup times. However, for arithmetic and calculations, all times must be converted to the base time units if they're not already in it. Arena allows you to express input times in different units, but you must also declare the base time units into which all times are converted during the simulation run, and in which all time-based outputs are reported.

The system starts at time 0 minutes with no parts present and the drill press idle. This *empty-and-idle* assumption would be realistic if the system starts afresh each morning, but might not be so great as a model of the initial situation to simulate an ongoing operation. The time durations that will make the simulation move are in Table 1. The (sequential) part number is in the first column, the second column has the time of arrival of each part, the third column gives the time *between* a part's arrival and that of the next part (called an *interarrival time*), and the service time (required to process on the drill press, not counting any time spent waiting in the queue) is in the last column. All times are in minutes. You're probably wondering where all these numbers came from; don't worry about that right now, and just pretend we observed them in the drilling center or that we brashly made them up.

Part Number	Arrival Time	Interarrival Time	Service Time
1	0.00	1.73	2.90
2	1.73	1.35	1.76
3	3.08	0.71	3.39
4	3.79	0.62	4.52

Table 1: *Arrival, Interarrival, and Service Times of Parts (in Minutes)*

We've decided that the simulation will stop at exactly time 20 minutes. If there are any parts present at that time (in service at the drill press or waiting in the queue), they are never finished.

Analysis Options

With the model, its inputs, and its outputs defined, you next have to figure out how to get the outputs by transforming the inputs according to the model's logic. In this section, we'll briefly explore a few options for doing this.

- Educated Guessing
- Queueing Theory
- Mechanistic Simulation

Queueing Theory

Since this is a queue, why not use queueing theory? It's been around for almost a century, and a lot of very bright people have worked very hard to develop it. In some situations, it can result in simple formulas from which you can get a lot of insight. Probably the simplest and most popular object of queueing theory is the *M/M/1 queue*. The first "M" states that the arrival process is *Markovian*; that is, the interarrival times are independent and identically distributed "draws" from an exponential probability distribution (see Appendices B and C for a brief refresher on

probability and distributions). The second "M" stands for the service-time distribution, and here it's also exponential. The "1" indicates that there's just a single server. So at least on the surface this looks pretty good for our model. Better yet, most of our output performance measures can be expressed as simple formulas. For instance, the average waiting time in queue (expected from a long run) is

$$\frac{\mu_S^2}{\mu_A - \mu_S}$$

where μA is the expected value of the interarrival-time distribution and μS is the expected value of the service-time distribution (assuming that $\mu A > \mu S$ so the queue doesn't explode). So one immediate idea is to use the data to estimate μA and μS , then plug these estimates into the formula; for our data, we get 3.462/(4.08 – 3.46) = 19.31 minutes. Such an approach can sometimes give a reasonable order-of-magnitude approximation that might facilitate crude comparisons.

The estimates of μA and μS aren't exact, so there will be error in the result as well.The assumptions of exponential interarrival-time and service-time distributions are essential to deriving the formula above, and we probably don't satisfy these assumptions. This calls into question the validity of the formula. While there are more sophisticated versions for more general queueing models, there will always be assumptions to worry about. The formula is for long-run performance, not the 20-minute period we want. This is typical of most (but not all) queueing theory. The formula doesn't provide any information on the natural variability in the system. This is not only a difficulty in analysis but might also be of inherent interest itself, as in the variability of production. (It's sometimes possible, though, to find other formulas that measure variability.) Many people feel that queueing theory can prove valuable as a first-cut approximation to get an idea of where things stand and to provide guidance about what kinds of simulations might be appropriate at the next step in the project. We agree, but urge you to keep in mind the problems listed above and temper your interpretations accordingly.

Pieces of a Simulation Model

We'll talk about the various parts of a simulation model in this session, all in reference to our example.

- **Entities**

Most simulations involve "players" called *entities* that move around, change status, affect and are affected by other entities and the state of the system, and affect the output performance measures.

Entities are the *dynamic* objects in the simulation—they usually are created, move around for a while, and then are disposed of as they leave. It's possible, though, to have entities that never leave but just keep circulating in the system. However, all entities have to be created, either by you or automatically by the software. The entities for our example are the parts to be processed. They're created when they arrive, move through the queue (if necessary), are served by the drill press, and are then disposed of as they leave. Even though there's only one kind of entity in our example, there can be many independent "copies," or *realizations* of it in existence at a time, just as there can be many different individual parts of this type in the real system at a time. Most entities represent "real" things in a simulation. You can have lots of different kinds of entities and many realizations of each kind of entity existing in the model at a time. For instance, you could have several different *kinds* of parts, perhaps requiring different processing and routing and having different priority; moreover, there could be several realizations of each kind of part floating around in the model at a time.

- **Attributes**

To individualize entities, you attach *attributes* to them. An attribute is a common characteristic of all entities, but with a specific value that can differ from one entity to another. For instance, our part entities could have attributes called Due Date, Priority, and Color to indicate these characteristics for each individual entity. It's up to you to figure out what attributes your entities need, name them, assign values to them, change them as appropriate, and then use them when it's time (that's all part of modeling). The most important thing to remember about attributes is that their values are tied to specific entities. The same attribute will generally have different values for different entities, just as different parts have different due dates, priorities, and color codes. Think of an attribute as a tag attached to each entity, but what's written on this tag can differ across entities to characterize them individually. An analogy to traditional computer programming is that attributes are *local* variables—in this case, local to each individual entity. Arena keeps track of some attributes automatically, but you may need to define, assign values to, change, and use attributes of your own.

- **Variables**

A *variable* (or a *global* variable) is a piece of information that reflects some characteristic of your system, regardless of how many or what kinds of entities might be around. You can have many different variables in a model, but each one is unique. There are two types of variables: Arena built-in variables (number in queue, number of busy servers, current simulation clock time, and so on) and user-defined variables (mean service time, travel time, current shift, and so

on). In contrast to attributes, variables are not tied to any specific entity, but rather pertain to the system at large. They're accessible by all entities, and many can be changed by any entity. If you think of attributes as tags attached to the entities currently floating around in the room, then think of variables as (rewriteable) writing on the wall. Variables are used for lots of different purposes. For instance, the time to move between any two stations in a model might be the same throughout the model, and a variable called Transfer Time could be defined and set to the appropriate value and then used wherever this constant is needed; in a modified model where this time is set to a different constant, you'd only need to change the definition of Transfer Time to change its value throughout the model. Variables can also represent something that changes *during* the simulation, like the number of parts in a certain subassembly area of the larger model, which is incremented by a part entity when it enters the area and decremented by a part when it leaves the area. Arena Variables can be vectors or matrices if it is convenient to organize the information as lists or two-dimensional tables of individual values. Some built-in Arena variables for our model include the status of the drill press (busy or idle), the time (simulation clock), and the current length of the queue.

- **Resources**

Entities often compete with each other for service from *resources* that represent things like personnel, equipment, or space in a storage area of limited size. An entity *seizes* (units of) a resource when available and *releases* it (or them) when finished. It's better to think of the resource as being given to the entity rather than the entity being assigned to the resource since an entity (like a part) could need simultaneous service from multiple resources (such as a machine and a person). A resource can represent a group of several individual servers, each of which is called a *unit* of that resource. This is useful to model, for instance, several identical "parallel" agents at an airline ticketing counter. The number of available units of a resource can be changed during the simulation run to represent agents going on break or opening up their stations if things get busy. If a resource has multiple units, or a variable number of units, we have to generalize our definition of resource utilization to be the time-average number of units of the resource that are busy, divided by the time-average number of units of the resource that are available. In our example, there is just a single drill press, so this resource has a single unit available at all times.

- **Queues**

When an entity can't move on, perhaps because it needs to seize a unit of a resource that's tied up by another entity, it needs a place to wait, which is the purpose of a *queue*. In Arena, queues have names and can also have capacities to represent, for instance, limited floor space for a

buffer. You'd have to decide as part of your modeling how to handle an entity arriving at a queue that's already full.

Comments

Lab Session No.3

To build simple process system model by using ARENA software.

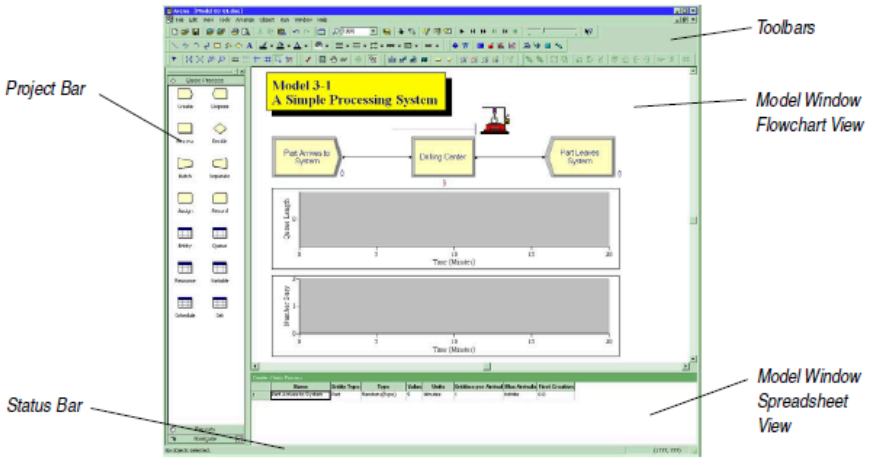
Theory

Exploring the Arena Window

In this session, we'll open an existing model, use it to look around the Arena window so you can get familiar with where things are, and introduce some basic Arena terminology.

Opening a Model

The ready-made model for the hand simulation can be found via *File* > *Open* (or just click to file open bring up the Open dialog box). File names appear in a scrolling box, and you can also navigate to other folders or drives.



Building Model 3-1 Yourself

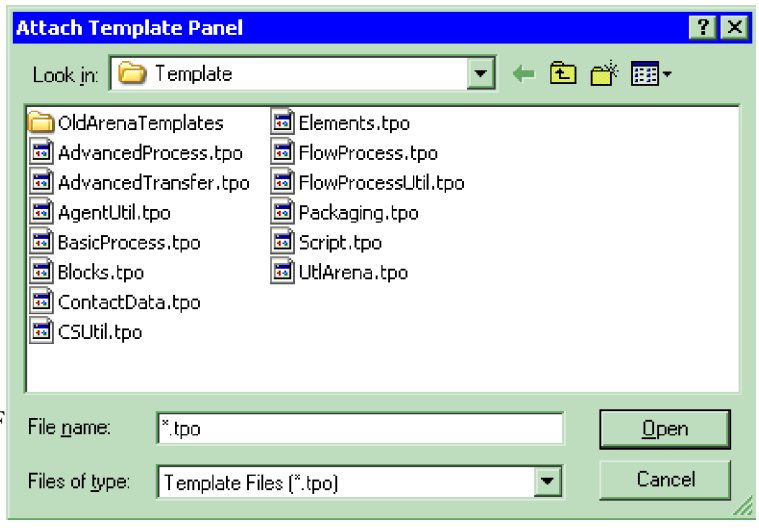
In this session, we'll lead you through the construction of Model 3-1 from scratch. What

you end up with might not look exactly like our Model 3-1 cosmetically, but it should be functionally equivalent and give you the same results. Before embarking on this, we might mention a couple of little user-interface functions that often come in handy: _ Right-clicking in an empty spot in the flowchart view of the model window brings up a small pop-up box of options, one of which is to repeat the last action (like placing a module in your model, of which you may need multiple instances).This can obviously save you time when you have repetitive actions (though it won't make them any more interesting). Other options here include some views, as well as running or checking the model.*Ctrl+D* or pressing the *Ins* key duplicates whatever is selected in the flowchart view of a model window, offsetting the copy a little bit. You'll then probably want to drag it somewhere else and do something to it.

New Model Window and Basic Process Panel

Open a new model window with (or *File* > *New* or *Ctrl+N*), which will automatically be given the default name Model1, with the default extension *.doe* when you save it. You can change this name when you decide to save the contents of the model window. Subsequent new model windows during this Arena session will get the default names Model2, Model3, and so on. You might want to maximize your new model window within the Arena window by clicking near the model window's upper-right corner (if it's not already maximized).

Next, attach the panels you'll need if they're not already there in the Project Bar. For this model, we need only the Basic Process panel, which is a file called BasicProcess.tpo, typically in the Template folder under the Arena folder. Clickm (or *File* > *Template Panel* > *Attach* or right-click on the Project Bar and select *Template Panel* > *Attach*) to open the Attach Template Panel dialog in Figure given below , where you open the Basic Process panel BasicProcess.tpo (click on it, then click the *Open* button, or just double-click it). You can tell Arena to attach certain panels automatically to the Project Bar of new model windows via *Tools* > *Options* > *Settings* by typing the panels' file names (including the *.tpo* extension) into the Auto Attach Panels box. The attached panel will appear in the Project Bar, with icons representing each of the modules in this panel. Right-click in the panel to change the icon size or to display text only for the modules. Right-clicking in a panel also allows you to detach it if you accidentally attached the wrong one (you can also detach the visible panel via or *File* > *Template Panel* > *Detach*). You can detach a panel even if you've placed modules from that panel in your model. If your display isn't tall enough to show the panel's full height, use the scroll bar at its right to get to it all.



Place and Connect the Flowchart Modules

This model requires one instance of each of three flowchart modules: Create, Process, and Dispose. To add an instance of a flowchart module to your model, drag its icon from the Project Bar into the flowchart view of the model window and drop it about where you want it (you can always drag things around later). To help you line things up, remember Grid ( ), Snap ( ), Snap to Grid ( ), Rulers ( ), Guides ( ), Glue ( ), *Arrange* > *Align*, and *Arrange* > *Distribute*. If you have *Object* > *Auto-Connect* checked and you dragged the modules into your model in the order mentioned above (without de-selecting a module after dropping it in), Arena will connect your modules in the correct order; if you have *Object* > *Smart Connect* checked, those connections will be oriented horizontally and vertically. Figure 5 shows how these modules look in the flowchart view of the model window just after we placed them (both Connect toggles were checked), with the Dispose model selected since it was the last one we dragged in. If you did not have *Object* > *Auto-Connect* checked, you'll need to connect the modules yourself; to do so, use *Connect* ( ) on the modules' exit ( ) and entry ( ) points, running along the connections if the *Animate Connectors* button ( ) is pressed (or equivalently, *Object* > *Animate Connectors* is checked) just to let you know that they're moving from one flowchart module to the next. However, this movement is happening in zero simulated time.

The Create Flowchart Module

Open the "raw" Create module by double-clicking it to get the dialog box in Figure 6 where we need to edit several things. We first changed the Name of this instance of the Create module from the default, Create 1, to Part Arrives to System, and we changed the Entity Type from the default to Part. In the Time Between Arrivals area of the dialog box, we accepted the default Random (Expo) for the Type, changed the default

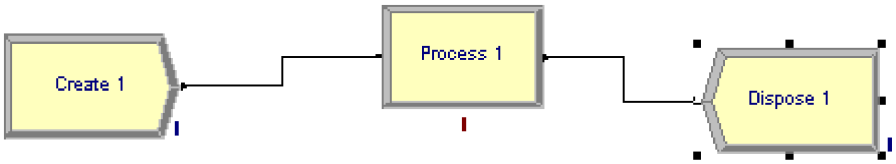


Figure 5: Initial placement of flowchart module

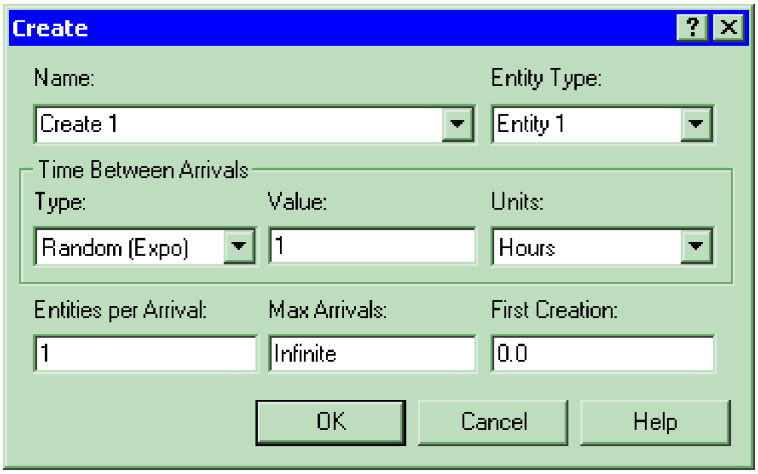
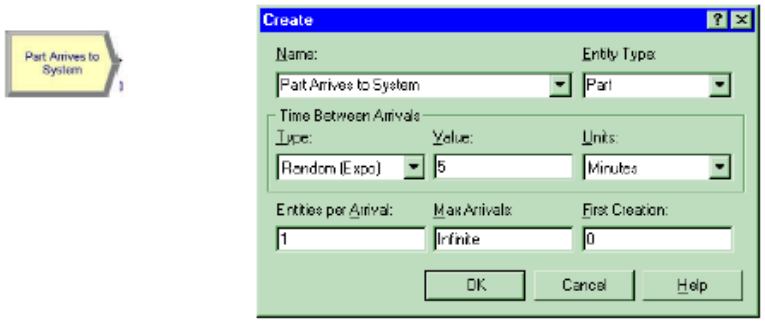


Figure 6: ***The Create Dialog Box***

Value of 1 to 5, and selected Minutes as the Units from the drop-down list there (note that the default is Hours). We accepted the defaults for the three boxes in the bottom row of the dialog box and clicked *OK* to save our changes; at this point, the Create dialog box should look like Figure 5. Recall that we can also view and edit flowchart modules via their spreadsheet view, as detailed in Section 3.3; the completed spreadsheet for this Create module was shown earlier in Figure 3-3. Note that the new name of this Create module now appears in its shape in the flowchart view of the model window.

Displays

As we introduce new modules and new concepts, we'll try to lead you through each dialog box (or, equivalently, spreadsheet). Even though the Create module is fairly simple, the above description was fairly lengthy. To convey this more compactly, we'll use visuals, called *Displays* .Note that there are three parts to this display. The upper-right portion has the filled-in dialog box . In some cases, it may show several related
dialogs. The upper left shows the module with which the dialog box is associated. Later, this portion may also show buttons we clicked to get the dialog box(es) shown on the upper right. The bottom portion of the display is a table showing the actions required to complete the dialog box(es). The left column of the table defines the dialog boxes or prompts, and the right column contains the entered data or action (italics) like checking boxes or pressing command buttons. In general, we'll try to provide the complete display when we introduce a new module or a new secondary dialog box of a module we've already covered. For modules that aren't new, we'll normally give you only the table at the bottom of the display, which should allow you to recreate all the models we develop easily. There may be more items or prompts in a dialog box than we show in a display; forthose we accept the defaults.



Name	Part Arrives to System
Entity Type	Part
Time Between Arrivals area	
Type	Random (Expo)
Value	5
Units	Minutes

Figure 7 ***Completed Create Dialog Box***

The Entity Data Module

One of the things we did in our Create module was to define an Entity Type that we called Part. By selecting the Entity data module in the Project Bar, the Entity spreadsheet for your model shows up in the spreadsheet view of the model window, as in Figure 9. Here you can see and edit aspects of the types of entities in your model. In Figure 9 the dropdown list for the Initial Picture field is shown, indicating that we decided that our Part entities will be animated as blue balls when the simulation runs. There are several fields for defining the costing data for entity types. A check box at the end lets you ask for Report Statistics on this entity type, including the average and maximum time in system observed for these types of entities during the run. We have only one entity type in our model, but if you had several, each would have its own row in the Entity spreadsheet.

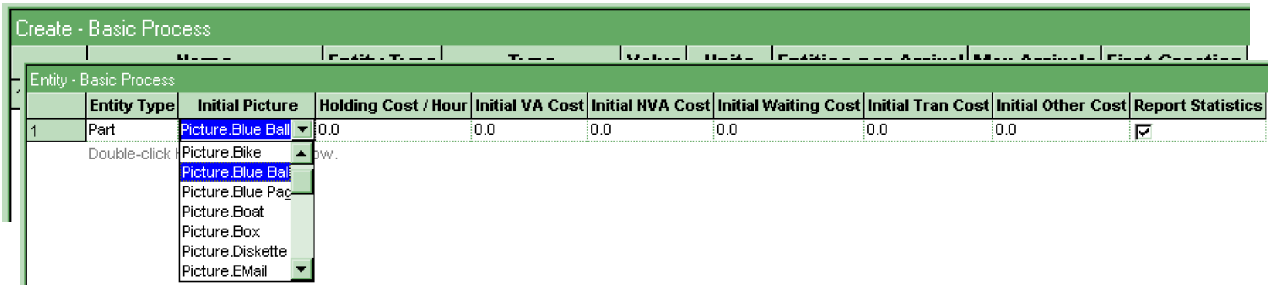
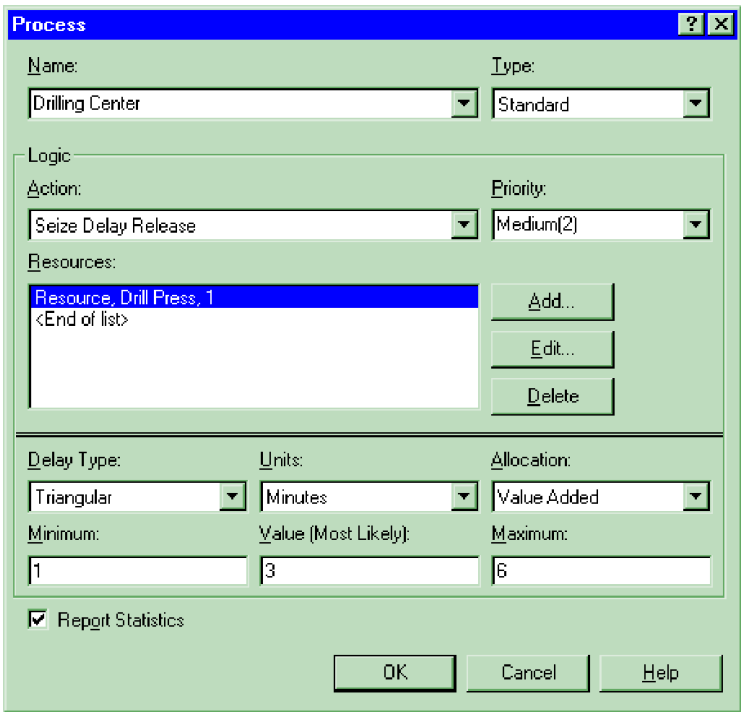


Figure 9: ***The Entity Spreadsheet for Model***

The Process Flowchart Module

Our Process module, which we named Drilling Center, represents the machine, including the resource, its queue, and the entity delay time there (part processing, in our case). Open it by double-clicking on its name, and you should see the dialog box in Figure 10. After entering the Name Drilling Center, we selected Standard as the type, meaning that the logic for this operation will be defined here in this Process module rather than in a hierarchical submodel. Skipping to the bottom of the dialog box, the Report Statistics check box allows you a choice of whether you want output statistics like utilizations, queue lengths, and waiting times in queue.The Logic area boxes take up most of the dialog and determine what happens to entities in this module.

Property Dialog Box



The Process for Model

Figure 10: ***The Resources Dialog Box for Model***

The Action we chose, Seize Delay Release, indicates that we want this module to take care of the entity's seizing some number of units of a Resource (after a possible wait in queue), then Delay for a time representing the service time, and then Release unit(s) of the Resource so that other entities can seize it. Other possible Actions are simply to Delay the entity here for some time (think of it like a red traffic light, after which the entity proceeds), Seize the Resource and then Delay (but not Release the Resource), or Delay and then Release the Resource that previously had been Seized; several Process modules could be strung together to represent a wide range of

processing activities. You can specify different Priorities for entities to Seize the Resource. Here and elsewhere in Arena, lower numbers mean higher priority. Define the Resource(s) to be Seized or Released in the Resources box; click *Add* to add a Resource to this list. You can define or edit a particular Resource line by double clicking its line, or selecting it and then clicking *Edit*, to bring up the Resources dialog box, as in Figure 11. Here you define the Resource Name and the Quantity of units (e.g., individual servers) that Seizing entities will Seize and that Releasing entities will Release (this is *not* where you specify the number of units of the Resource that exist—that's done in the Resource data module's Capacity field and will be discussed later). Listing more than one Resource means that Seizing entities must Seize the specified Quantity of each Resource before starting to be processed, like a machine and two operators, and Releasing entities will Release the specified Quantity of the corresponding Resources. Returning to the Process dialog box .the Delay Type drop-down list box offers three probability distributions (Normal, Triangular, and Uniform), a Constant, or a general Expression. The Units field determines the time units for the numerical Delay duration, and the Allocation field relates to how this delay is to be charged. The prompts on the next line change to match your choice of Delay Type. Note that the Expression option for the Delay Type allows you great flexibility in defining the Delay duration, including any other Arena probability distribution; right-clicking in the Expression field lets you bring up the Expression Builder to help you.

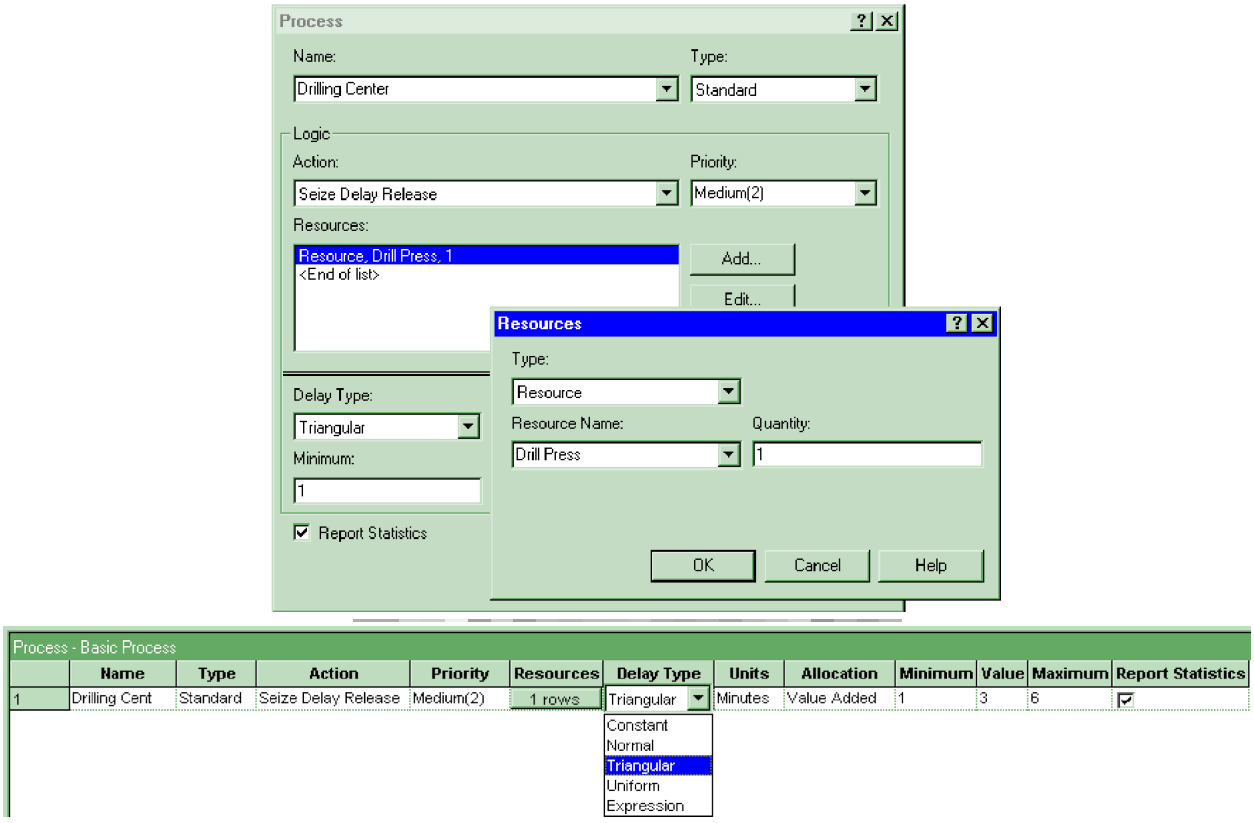
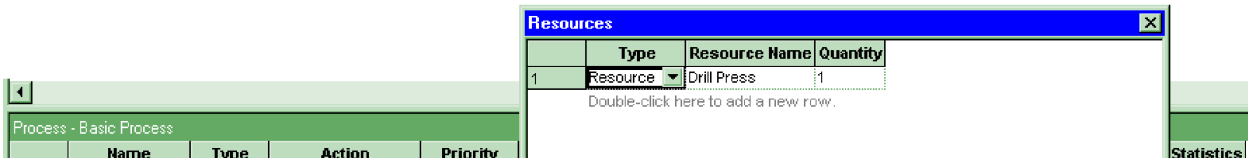


Figure 12:The Process Spreadsheet for Model

Figure 13:***The Resources Secondary Spreadsheet in the Process Spreadsheet for Model***

CS



Close the Process dialog box with the *Cancel* button; again, if you had made changes that you wanted to retain, you'd click *OK*. Figure 12 illustrates the Process spreadsheet, seen if you select any Process module instance in the flowchart view of the model window, or the general Process module in the Project Bar, with the drop-down box for Delay Type shown (where we've selected Triangular). If you had multiple Process modules in your model, there would be a row for each one in the Process spreadsheet. As with Create modules, this provides an alternative way to view simultaneously and edit the fields for your Process module(s). If you click on the "1 Rows" button in the Resources field, a secondary spreadsheet appears (see Figure 13) that allows you to edit, add, and delete resources equivalent to the Resources dialog box of Figure 11 (you need to click the cross button at the upper right of the Resources secondary spreadsheet to close it before you can go on).

The Resource Data Module

Once you've defined a Resource as we've done in this Process module (in our case, we named the resource Drill Press), an entry for it is automatically made in the Resource data module; click on it in the Project Bar to view the Resource spreadsheet in Figure 14 This spreadsheet allows you to determine characteristics of each Resource in your model, such as whether its Capacity is fixed or varies according to a Schedule (that drop-down list is shown in Figure 13, where Fixed Capacity has been selected). You can also cause the Resource to fail according to some pattern; try clicking on the "0 rows" button under the Failures column heading to bring up a secondary spreadsheet for this (the failure pattern is defined in the Failure data module in the Advanced Process panel, which you might have to attach to the Project Bar for your model).

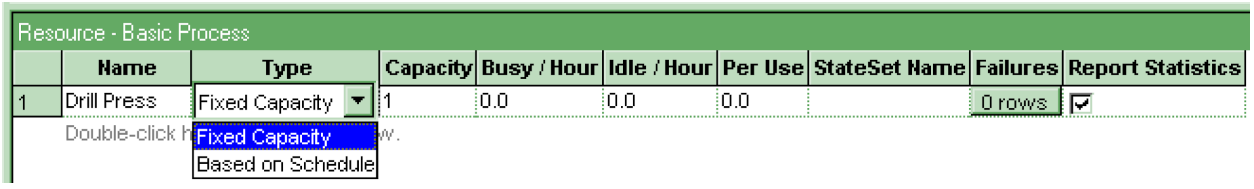


Figure 14: *The* ***Resource Data Module Spreadsheet for Model***

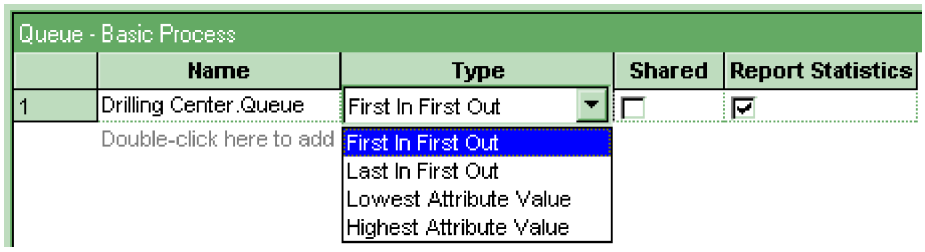


Figure 15: ***The Queue Data Module Spreadsheet for Model***

The Queue Data Module

If the Drill Press resource is busy when an entity gets to the Process module, the entity will have to queue up. The Queue spreadsheet, seen in Figure 15, appears in the spreadsheet view if you select the Queue data module in the Project Bar. Here you can control aspects of the queues in your model (we have only one, named Drilling Center.Queue), such as the discipline used to operate it, as shown in the Type list in Figure 15 (First In First Out is the default and is selected). You could, for instance, rank the queue according to some attribute of entities that reside in it; if you chose Lowest Attribute Value, the queue would be ranked in increasing order of some attribute, and an additional field would show up in the line for this Queue in which you would have to specify the Attribute to be used for ranking.

Animating Resources and Queues

Speaking of queues, you might have noticed the just above the Process module in the flowchart view. This is where the queue will be animated, and the Process module acquired this graphic when we specified that we wanted entities to Seize a Resource there. And while we're on the subject of animation, you've no doubt noticed the  above and to the right of the Process module and positioned at what will be the head of the queue animation. This is a Resource animation and will change appearance during the simulation depending on whether the Drill Press Resource is Idle or Busy. This did not come "free" with the Resource specified in the Process module; rather, we added it to our model via the *Resource* button ( ) in the *Animate* toolbar. Double-click on the icon to get the Resource Picture Placement dialog, This allows us to pick pictures from libraries (files with extension *.plb* to their name, usually found in the Arena folder) to cause the Resource to be animated differently depending on the state it's in.

Dispose Flowchart Module

The Dispose module represents entities leaving the model boundaries; double-click its name to bring up the dialog box in Figure 17; the Dispose spreadsheet is in Figure 18. There's not much to do here—just give the module a descriptive Name and decide if you want output on the Entity Statistics, which include things like average and maximum time in system of entities that go out through this module and costing information on these entities.

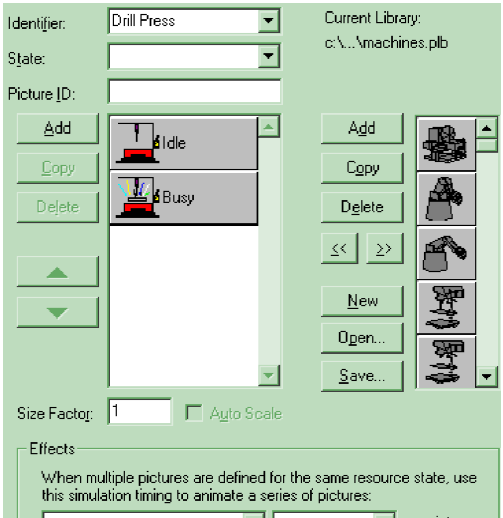


Figure 16: ***The Resource Picture Placement Dialog Box for Model***

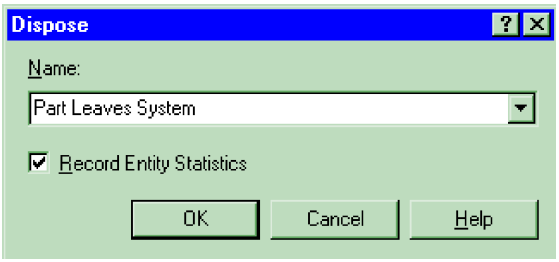


Figure 17: ***The Dispose Property Dialog Box for Model***

Figure 19: The Dispose Spreadsheet for Model

Connecting Flowchart Modules

The Create, Process, and Dispose modules are connected (in that order, going left to right) by lines called *Connections*. These establish the sequence that all parts will follow as they progress from one flowchart module to another. To make the Connections, click *Connect* ( ) or equivalently select *Object* > *Connect*, which changes the mouse pointer to cross hairs. Click on the *exit point* ( ) from the source module and finally on the *entry point* ( ) on the destination module (you can make intermediate clicks if you want this connection to be a sequence of line segments). To help you hit these exit and entry points, which might appear quite small if you're zoomed out to a high altitude, Arena lights up a green box to indicate that you can click now and hit an exit point, and a red box for an entry point. If you have many connections to make (maybe you placed a lot of flowchart modules to rough out your model), after each one you can right-click in a blank spot of the flowchart view and select Repeat Last Action from the pop-up menu to keep connecting. And if you have *really* a lot of connections to make, you can double-click on the Connect button (or do *Object* > *Connect* twice in a row) and not even have to bother with the
right-click pop-up; when you're done and you want to get out of this, right-click or hit the

Escape key (*Esc*). If *Object* > *Auto-Connect* is checked, Arena will automatically connect the entry point on a newly placed module to whichever other connect-out module is selected when you place the new module. If *Object* > *Smart Connect* is checked, then new Connections are automatically laid out to follow horizontal and vertical directions only, rather than following free-form diagonal directions according to where the connected modules are (unless you make intermediate clicks while drawing a connection, in which case you get the intermediate points and diagonals). This is pretty much a matter of taste and has no bearing on the model's operation or results.

If *Object* > *Animate Connectors* is checked (or, equivalently, *Animate Connectors* ( ) is pushed in), then Arena will show entity icons (in our case, the blue balls) running down the connections as the transfers happen when the simulation runs. This is just to let you know during the animation that these transfers are occurring—as far as the simulation and statistics collection are concerned, they are happening in zero simulated time (or, equivalently, at infinite speed). We'll show you how to model non-zero travel times between model locations in Section 4.4, including how they're animated. If for some reason you want to move a connection point (entry or exit) to somewhere else relative to its module, you can do so but you must first right-click on it and select Allow Move from the pop-up.

Setting the Run Conditions

Things like run length and number of replications are set via *Run* > *Setup*, which brings up a dialog box with five tabbed pages. Figure 3-15 shows the tab for Project Parameters, where we specify a Project Title, Analyst Name, and Project Description, as well as select what kind of output performance measures we want to be told about afterwards. We also chose to document our model internally via entering a brief Project Description.

Figure 20 shows the *Replication Parameters* tab of Run Setup, which controls a number of aspects about the run(s). We default the Number of Replications field to 1. We'll default (that is, not use) the Start Date and Time field, which is for associating a specific calendar date and time with a simulation time of zero. You can also specify a Warm-up Period at the beginning of each replication, after which the statistical accumulators are all cleared to allow the effect of possibly atypical initial conditions to wear off. We specify the Length of Replication to be 20 and select the time unit for that number to be Minutes. The Hours Per Day box defaults to 24 (to answer the obvious question you have about this, it could be convenient to define a day to have, say, 16 hours in the case of a two-shift manufacturing operation, if it's customary to think of time in days). The Base Time Units box specifies the "default" time units in which time-based outputs will be reported, as well as how Arena will interpret some time-based numerical inputs that don't have an accompanying Time Units box (such as Maximum in the Plot dialog under Axes > Time (X) Axis > Scale). The

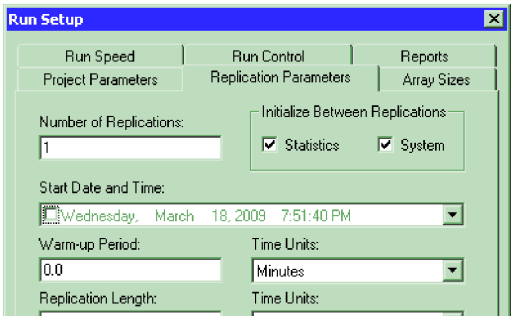


Figure 20: *The Run* > *Setup* > *Replication Parameters Dialog Box for Mode*

Terminating Condition box allows you to establish complex or state-dependent termination rules; for an example where we want the simulation to keep running until the results achieve the statistical precision we'd like. Model 3-1, however, will simply terminate at time 20 minutes. Close the Run Setup dialog box by pressing *Cancel*. Speaking of termination, you must specify in every Arena model how you want it to terminate. This is really part of modeling. Arena can't know what you want, so does not include any kind of "default" termination. In fact, in most cases, your simulation will just continue running forever or until you intervene to stop it, whichever comes first.

Running It

To run the model, click the *Go* button ( ) in the *Standard* toolbar (or *Run* > *Go* or press the *F5* key); note that the buttons in this group are similar to those on a video player. The first time you run a model (and after you make changes to it) Arena checks your model for errors (you can do this step by itself with the √ button on the *Run Interaction* toolbar, or *Run* > *Check Model* or the *F4* key); if you have errors, you'll be gently scolded about them now, together with receiving some help on finding and correcting them. Then you can watch the model animation run, but you'll have to look fast for a run this short unless your computer is pretty laid back. During the animated run, you see the Part entities (the blue balls) arriving and departing, the Resource Picture changing its appearance as the Resource state changes between Idle and Busy, the Queue changing as Part entities enter and leave it, the digital simulation clock in the Status Bar advancing, and the plots being drawn. The counters next to the flowchart modules display different quantities depending on the module type. For the Create module, it's the number of entities that have been created. For the Process module, it's the number of entities that are

currently in process there (in service plus in queue), and for the Dispose module, it's the number of entities that have left the system.

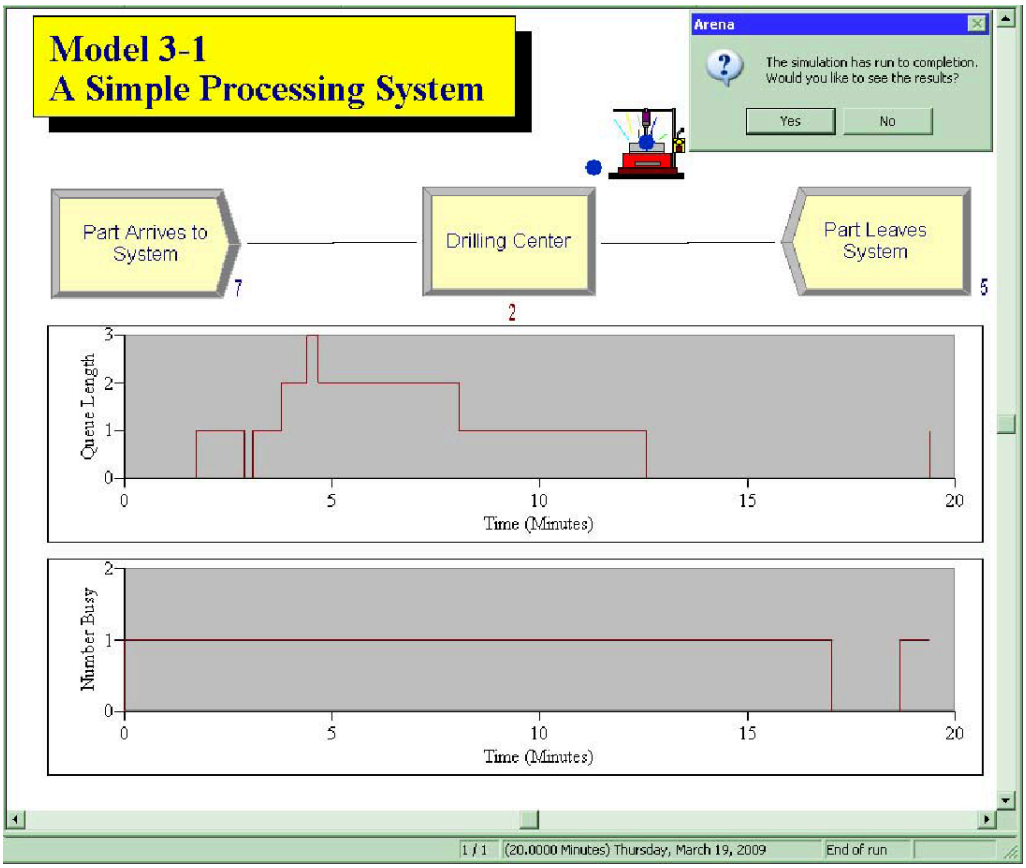
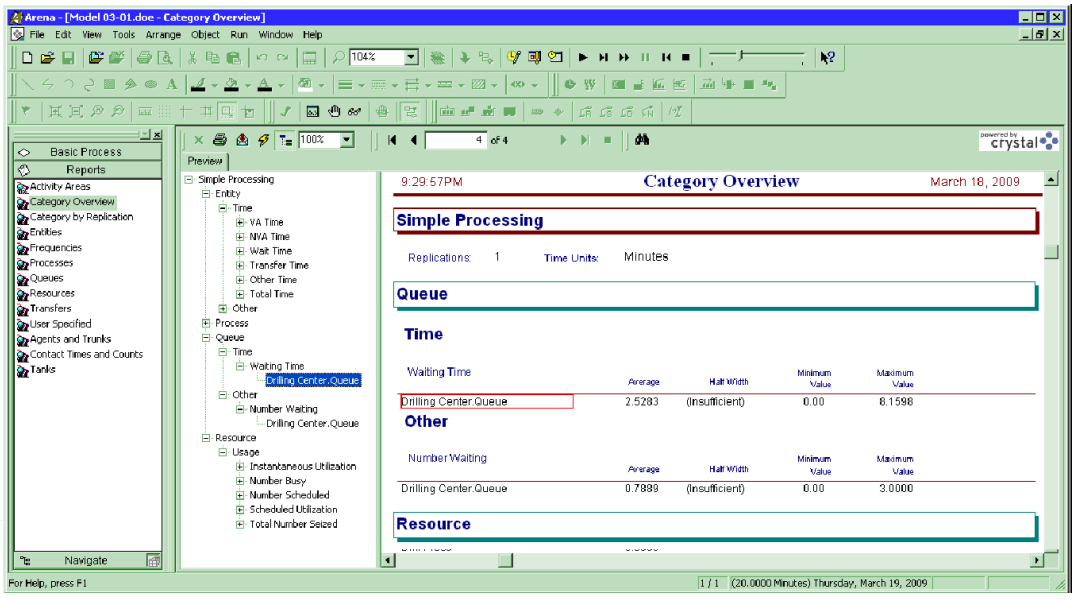


Figure 21;Ending Animation State of Model

Viewing the Reports



CS3

Right now you might be thinking that this report structure is pretty serious overkill just to get a handful of numbers out of this small model, and it probably is. However, in large, complicated models it's quite helpful to have this structure to organize the myriads of different output numbers and to help you find things and make some quick comparisons and conclusions.

Comments

Lab Session No.4

To build generalizes series and parallel loan application model by using ARENA.

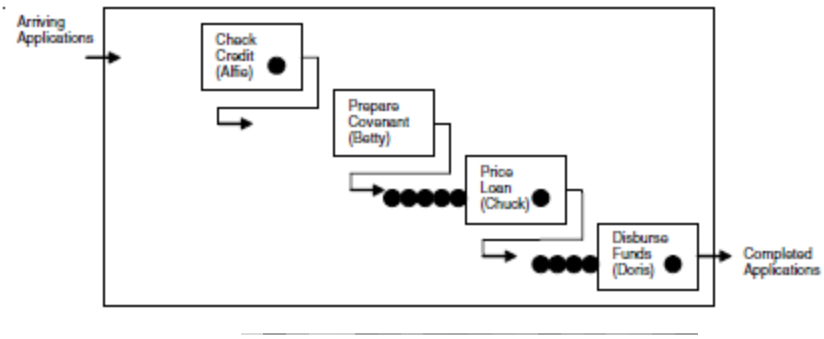
Theory

Model building Example

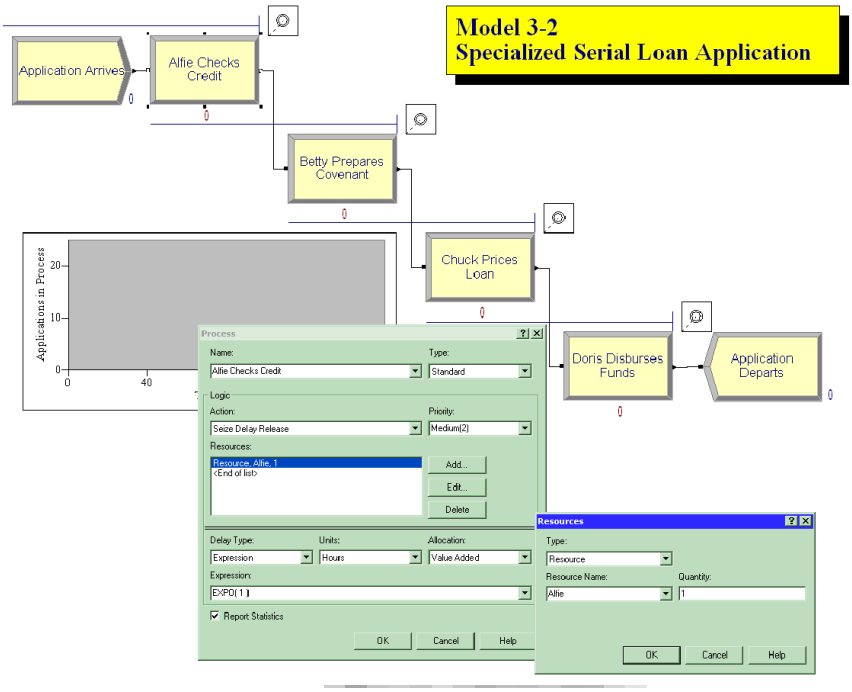
Consider a loan application office (as did Harrison and Loch), where applications arrive with exponentially distributed interarrival times with mean 1.25 hours; the first application arrives at time zero. Processing each application requires four steps: first a credit checks (this takes time but everyone passes), then preparing the loan covenant, then pricing the loan, and finally disbursement of funds. For each application, the steps have to be done in that order. The time for each step is exponentially distributed with mean 1 hour, independent of the other steps and of the arrival process. Initially, the system is empty and idle, and we'll run it for 160 hours (about a work month); see Chapter 5 for other kinds of stopping rules based on entity counts and other conditions. Output performance measures include the average and maximum total number of applications in process, and the average and maximum total time, from entry to exit, that applications spend in the system, as well as their time waiting for the next processing step to begin. There are four employees available (Alfie, Betty, Chuck, and Doris), all equally qualified for any of the four steps, and the question is how best to deploy them.

Serial Processing – Specialized Separated Work

A first thought might be to specialize the employees and assign, say, Alfie to check credit for all applications, Betty to prepare all covenants, Chuck to price all loans, and Doris to disburse all funds. So each application would first have to go through Alfie, then Betty, then Chuck, and finally Doris. A layout might look like Figure 22, where there are currently 12 applications (the filled circles) in process, Alfie is busy but with no more applications in queue, Betty is idle (so of course with no queue), Chuck is busy with five more in queue, and Doris is busy with four more in queue. All queues are FIFO. Though Betty might disagree, it's too bad that, under these operational rules, she can't lend a hand to Chuck or Doris right now.
An Arena simulation of this, Model 3-2, is fairly straightforward, and essentially just involves adding three Process modules to Model1, along with three more Resources. The completed model is in Figure 23, with the first Process module and its Resources dialog open.



Since the modules are so similar to those in Model 3-1, we'll just highlight the differences, and will trust you to browse through the model: The Create module is the same as in Model 3-1, except for the module Name, the name for the Entity Type, the Value of the mean of the exponential interarrival times, and the time Units (now Hours). The four Process modules have, as before, the Action of Seize Delay Release, but of course have different names for the modules and Resources. Since exponential process times are not among the Delay Type choices, instead choose Expression there, and then use the Expression Builder (go to "Random Distributions") to fill in the Expression field, as seen in Figure 23. The time Units here are also Hours.
Other than its Name, the Dispose module is the same as we take it.



- The default Entity Picture (Entity data module, Picture.Report) is okay here since the entities are, well, reports of the loan applications.
- _ The default Resource animations are almost okay, but we made the Idle picture also have a white fill rather than the default green (double-click on the Resource animation icon to get to the Resource Picture Placement window, then doubleclick on the Idle and Busy icons in turn and use the Fill Color tool). Remember to select in the Identifier field the correct Resource name for each of the four cases. Also, check the Seize Area box.
- _
- The Queue and Resource data modules are automatically filled in correctly if the Process modules were set up first. The defaults for everything (e.g., the Type and Capacity of each of the four Resources) work in this model.
- _ The dynamic Plot traces the total number of applications in process, which is the Arena Expression EntitiesWIP(Application), and can be found in the Expression Builder under Basic Process Variables, then Entity, and finally Number in Process. "WIP" is an acronym for work in process. The other entries in the Plot dialog are similar those in Model 3-1, adjusted of course
- for the different time frame and maximum y-axis value (we decided on a fixed
- maximum for comparison across this and the next three models).

In *Run* > *Setup*, in the Project Parameters tab check the Processes box in Statistics Collection, and enter some documentation in the three text fields; in the Replication Parameters tab enter 160 for the Replication Length and make sure the Time Units and Base Time Units are both Hours.

- After some initial runs, we extended the queue animations a bit out to the left to accommodate the maximum queue lengths. Run this model and look at the Category Overview report to find, among other things:
- The average and maximum total number of applications in process were, respectively, 12.3931 and 21 (via Loan Application → Entity → Other → WIP).

- The average and maximum total time, from entry to exit, that applications spent in the system were, respectively, 16.0831 hours and 27.2089 hours (Loan Application → Entity → Time → Total Time). Note that this counts only those applications that had left the system when the simulation stopped, not those that were still in process at that time (since, of course, their total time in system had not yet been completed).

- The average and maximum total time that applications spend waiting for the next processing step to begin were, respectively, 11.9841 hours and 22.2732 hours (Loan Application → Entity → Time → Wait Time). This includes only time "wasted" waiting in queue, and not "value-added" time spent undergoing processing at the four steps, so is a good measure of system inefficiency. Arena counts in this statistic the total waiting time in the four queues only for those
- applications that completed all four steps and exited the system; waiting times in the individual queues are under Loan Application → Queue → Time .
- During the 160 hours, 117 applications were completed (Loan Application →
- Entity → Other → Number Out), a measure of productivity Alfie, Betty, Chuck, and Doris were busy, respectively, 82.33%, 70.34%, 80.44%,
- and 80.80% of the time.

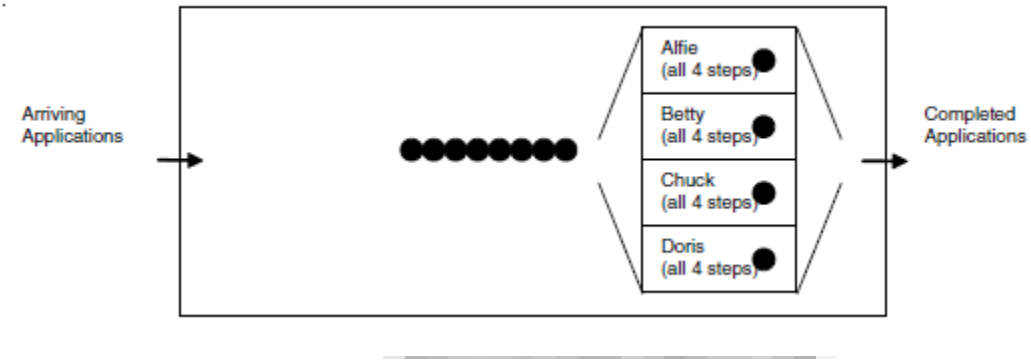
- Alfie, Betty, Chuck, and Doris processed, respectively, 128, 128, 122, and 117 applications (Loan Application → Process → Other → Number Out). In this model, all applications visit these people in this order, so these numbers in this order must decrease or stay the same. (Under what conditions would they all be the same? Is that possible in this model?)

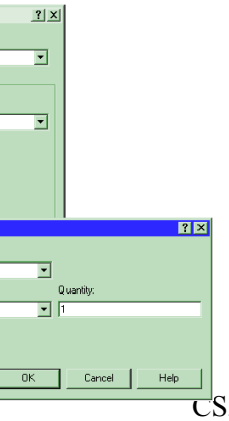
- The main inefficiency in this system is the waiting in queue that the applications must endure, and there are four different places where that could happen. Also, there is the possibility, as in Figure 22, that applications could be queued at some stations while other employees are idle, which seems like a waste of resources

Parallel Processing

Would it be better to "generalize" or "integrate" the work so that each employee completely processes all four steps of one application at a time, and with a single queue of applications

"feeding" the group? From the applications' viewpoint, this would present just one opportunity for wasting time waiting in queue, but then again the processing would be longer. Figure 3-27 shows how this would work, where each employee processes all four steps before moving on to the next application in queue. Like the specialized serial processing snapshot in Figure 22, there are currently 12 applications in process, but note that the total number in queue is now eight rather than nine, since Betty is now busy (we don't know how she feels about that, but this looks like better service). The Arena model for this, parallel Model ,is actually simpler than Model series, and is shown in its completed form in Figure 25 with the (sole) Process module and its Resources dialog showing. The Create and Dispose modules are identical to Model 3-2(Series), as is the dynamic Plot and the *Run* > *Setup* dialog (except for labeling).





The main change is that the four Process modules in Model 3-2, each of which represented a single employee, are replaced by a single Process module representing all four employees. We replaced the four single-unit resources in Model 3-2 with a single four-unit resource here, which we named Loan Officer (look at the Resource data module in Model pralle). Note that, in the Resources dialog of the Process module shown in Figure 25, the Quantity field is still 1, representing the number of units of the Loan Officer resource that are seized and released by each Application entity; this is *not* where we specify that there are four units of this resource in existence, but rather in the Capacity column of the Resource data module. And since each employee must now complete all four tasks in tandem before taking the next application out of the queue, we must make the Delay time longer in the Process module. Each of the four steps requires EXPO(1) hours to complete, so we simply add four of these up in the Expression field in the Delay Type area of the Process module, as seen in Figure 3-28, each representing one of the four processing steps (it's okay to do arithmetic inside an Expression field anywhere in Arena). Now we're guessing that, if you're reading a book like this, you probably did well in seventh-grade algebra, so you know that adding up four of the same thing is the same as multiplying that thing by 4, so you may be wondering why we didn't save ourselves some typing and enter something like 4*EXPO(1) instead. Well, the problem with that is that it would make just a single "draw" from the exponential distribution and multiply that single resulting number by 4, meaning in our case that each of the four steps took exactly the same amount of time, not what we want. So, ugly as it is, EXPO(1) + EXPO(1) + EXPO(1) + EXPO(1) is what we need since this makes four separate, independent draws from the exponential distribution to represent four separate, independent times for the four steps.

Comments

Lab Session No.5

To develop monte carlo simulation for a drilling center.

Theory

Introduction

With Arena, you can:

- Model your processes to define, document, and communicate.
- Simulate the future performance of your system to understand complex relationships and identify opportunities for improvement.
- Visualize your operations with dynamic animation graphics.
- Analyze how your system will perform in its "as-is" configuration and under possible "to-be" alternatives so that you can confidently choose the best way to run your business.

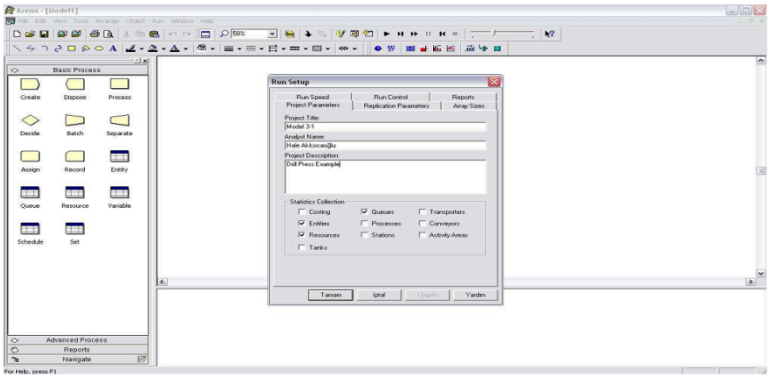
Our Task

We'll start with a simple case. In this manufacturing system, parts arrive exponentially with mean 5. If a part arrives and finds the drill press idle, its processing at the drill press starts right away; otherwise, it waits in First-In First-Out (FIFO) queue. Processing time for drill

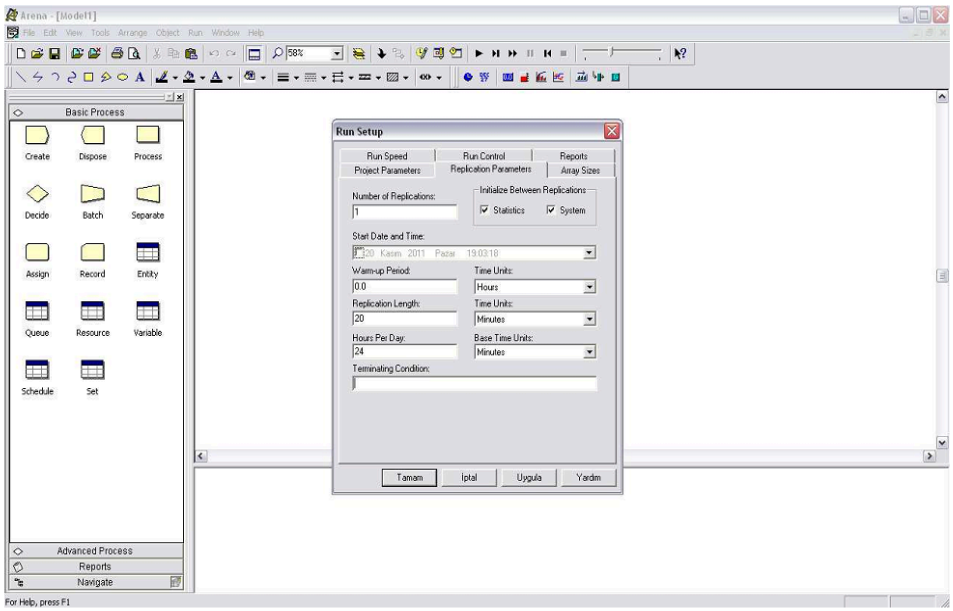
press operation is TRIA(1,3,6). After completion of drilling process, parts leave the system. Run the simulation for 20 minutes.

Building Model

Since you'll start to build a model, first you should define your project and running conditions. In order to do this, first select Run/Setup menu option. Then click on Project Parameters tab. Enter Project Title, Analyst Name and Project Description. Select what kind of output performance measures we want to be told about afterwards from Statistics Collection as shown in Figure below. For this example, the queue, resource and entity statistics will be enough.

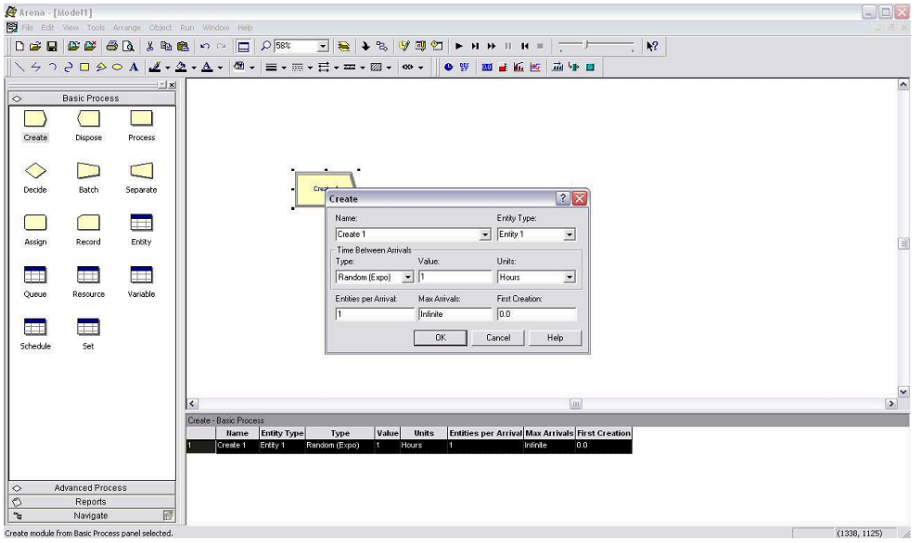


The Replication Parameters tab of Run/Setup, which controls a number of aspects about the run. First we need to enter Number of Replication. If we request multiple replications, we need to tell Arena what to do between replication. Since we request single replication, we do not make any changes "Initialize Between Replications" part. In this case, system has no warm-up period but in many systems, operations cannot start with doors opening. Then enter replication length and its time unit and if we have any terminating condition, we should enter this information into Terminating Condition box. The completed tab for this example is shown in Figure.

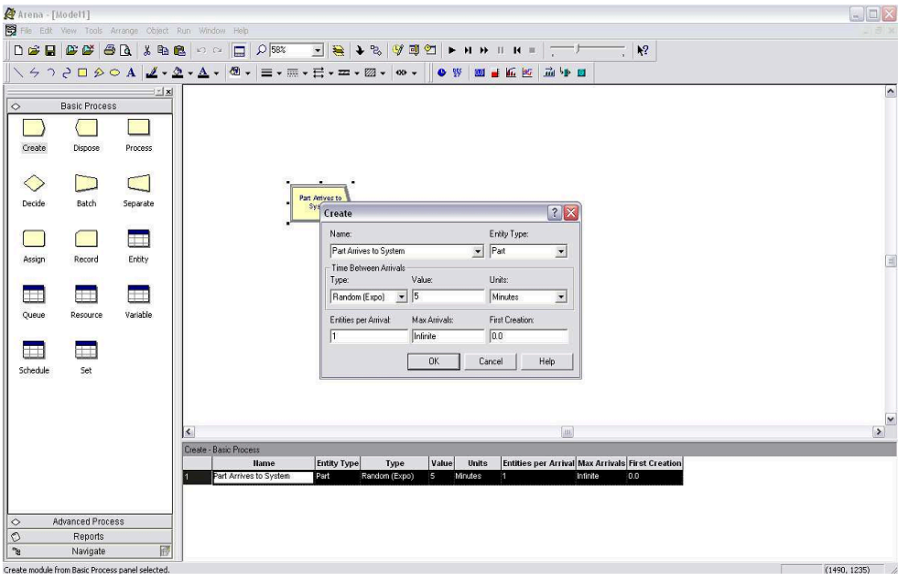


Create Flowchart Module

Click on Create from Basic Process panel and drag and place it into Flowchart View. Open Create module by double-clicking it to get dialog box in Figure.

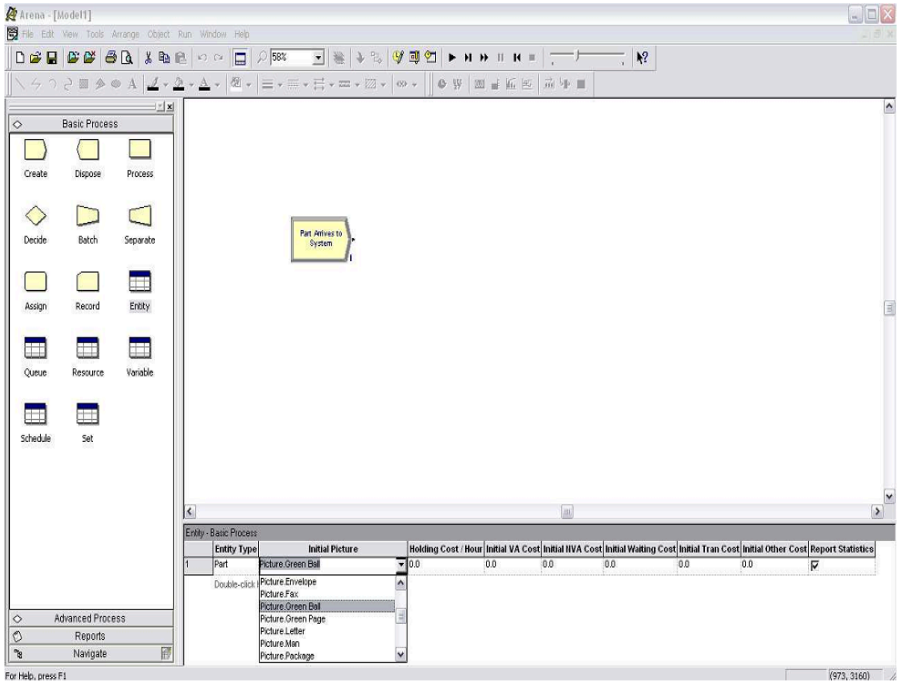


First change the name of module, Entity Type. In the Time Between Arrivals area of the dialog box, we accepted the default Random (Expo) for the Type, changed the default Value of 1 to 5, and selected Minutes as the Units from the drop-down menu. Finally click OK to save changes and dialog box should look like Figure.



Entity Data Module

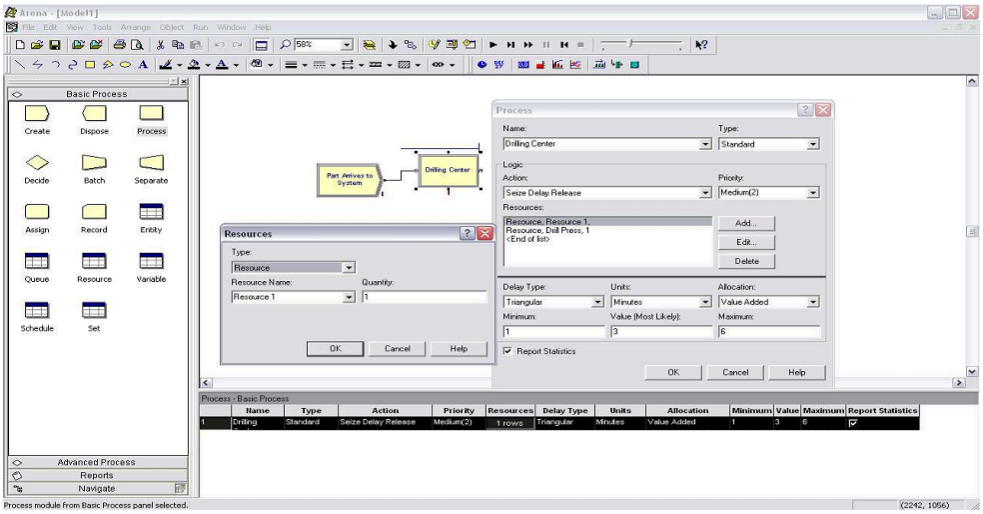
Previously, we define Entity Type as Part in Create module. But you might want to change some things about it. To do this, click the Entity data module in Basic Process panel. For instance you can change animation picture for Parts as Figure.



Process Flowchart Module

Click on Process module in Basic Process panel and drag it into flowchart view. Double click on it and edit Process module. After entering name of the process, you need to select type of action. There are four action options.

- Delay: Simply indicates that a process delay will be incurred with no resource constraint.
- Could just *Delay* entity (red traffic light) – no Resources or queuing
- Seize Delay: It indicates that a resource(s) will be allocated in this module and a delay will occur, but that resource release will occur at a later time.
- Could also *Seize Delay* (no Release … Release downstream)
- Seize Delay Release: This option indicates that a resource(s) will be allocated followed by a process delay and then the allocated resource(s) will be released. We use this option for FIFO queuing discipline
- *Seize Delay Release* – entity Seizes some number of units of a Resource (maybe after a wait in queue), Delay itself there for the processing time, then Release the units of the Resource it had Seized – we chose this option.
- Delay Release: This indicates that a resource(s) has previously been allocated and that the entity will simply delay and release the specified resource(s).
- If you select options except first one you should also add a resource or resources to seize and/or release.
- Then you should define Delay Type, Units and related values. The completed Process module for this model is shown in Figure.



Resource And Queue Data Module

Once you've defined this process module, your model has both a Resource and a Queue, with names you specified for resource and default name of Process Name. Queue for queue. If you want to change default name you can change it from spreadsheet view of Resource or Queue module.

Dispose Flowchat Module

The final flow chart module is the Dispose.

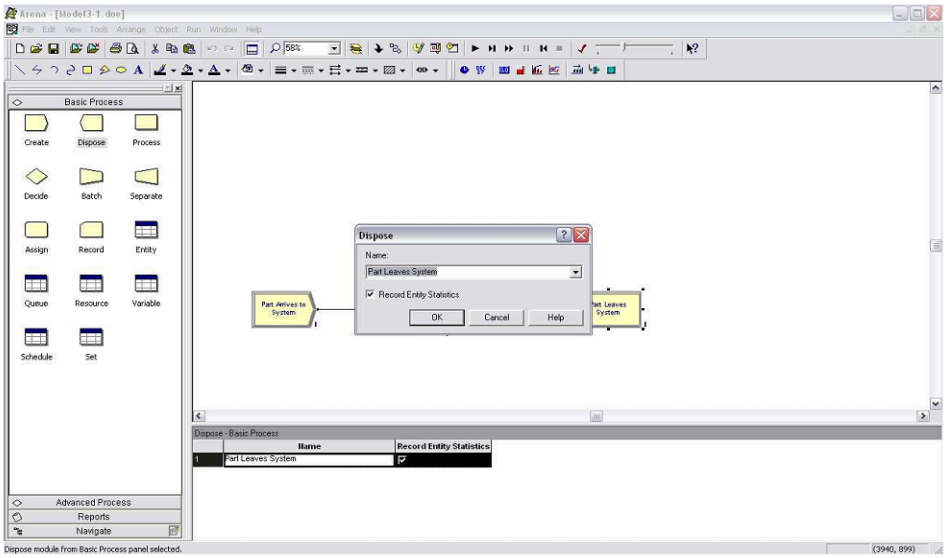


Figure 12. Dispose Module Dialog Box

RUNNING IT

To Run the model, click the Go button in the Standard toolbar (or Run >> Go or press the F5 key); note that the buttons in this group are similar to those on a video player. The first time you run a model Arena checks your model for errors (or you can do this step by itself with the button on the run Interaction toolbar, or Run >> Check Model or the F4 key); if you have errors, you'll gently scolded about them now, together with receiving some help on finding and correcting them.

The Arena box that appears at the end of the run asks if you'd like to see the summary results. After you look at those reports (or if you choose not to), your model window will appear to be hung and you can't edit anything. That's because you're still in run mode for the model, which gives you a chance to look at the plots and the final status of the animation. To get out of run mode and back to being able to edit, you have to click End , just like on a video player.

Viewing the Reports

If you'd like to see the numerical results now, click Yes in the Arena box that appears at the end of your simulation. This opens a new reports window in the Arena window. The Project Bar now displays the Reports panel, which lists several different Reports you can view, such as Category Overview, Category by Replications, and Resources. Clicking on each of these reports in the Project Bar opens a separate report window in the Arena window. Don't forget to close these report windows when you're done viewing them since they don't go away on their own if you simply go back to your model window; if you change your model then re-run it.

Lab Session No.6

To Simulate and analyze a bank problem.

Theory

Continuous vs Discrete:

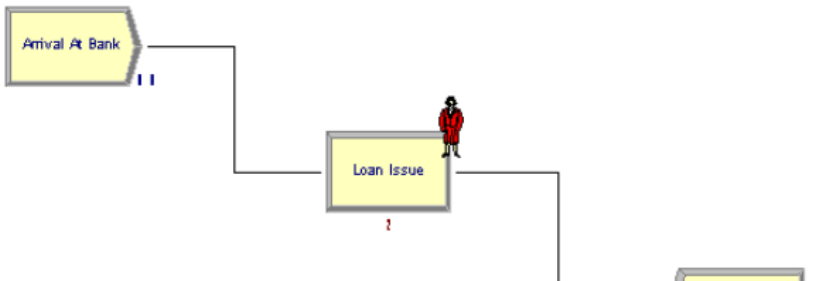
- In Continuous model state of the system can change continuously over time. E.g. Levels of a water reservoir falls due to evaporation occur.
- In a Discrete model change can occur only at separated points in time. E.g. A manufacturing system with parts arriving and leaving at specific time
- Arena is mostly focused in discrete models.

Deterministic vs stochastic:

- Model that have no random input are deterministic.
  E.g. Strict appointment-book with fixed service time.
- Stochastic models operate with at least some inputs being random.
  E.g. A bank with randomly arriving customer requiring varying service times.

Procedure:

At first use the create model to set number of customer coming inside the bank system.Than use process module to process customer coming inside the system for loan issue and finally use the dispose module to discard customer when processed.



Lab Session No.7

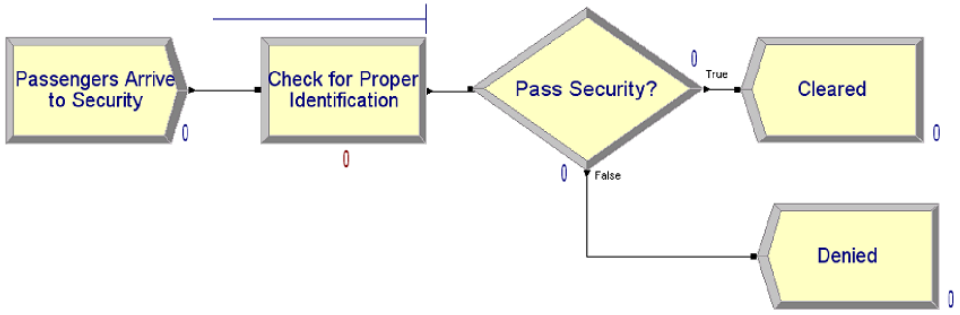
To Simulate and analyze an airport security system.

Theory

OUR TASK

Analyze and Simulate a simple airport security line to illustrate how you can model, simulate, visualize and analyze with Arena Software.

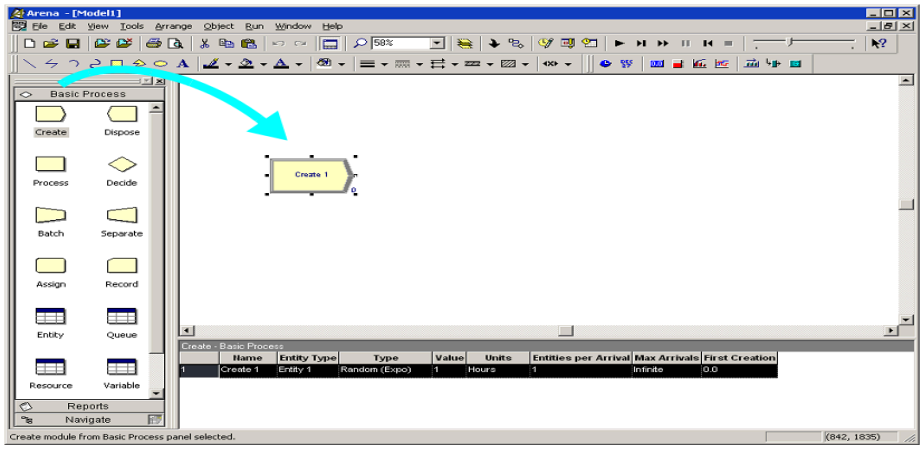
process of passengers arriving at security for identification inspection. We will build the flowchart shown below, introducing you to the process of modeling and simulating with Arena.



CREATE THE SECURITY IDENTIFICATION REVIEW ENTITIES

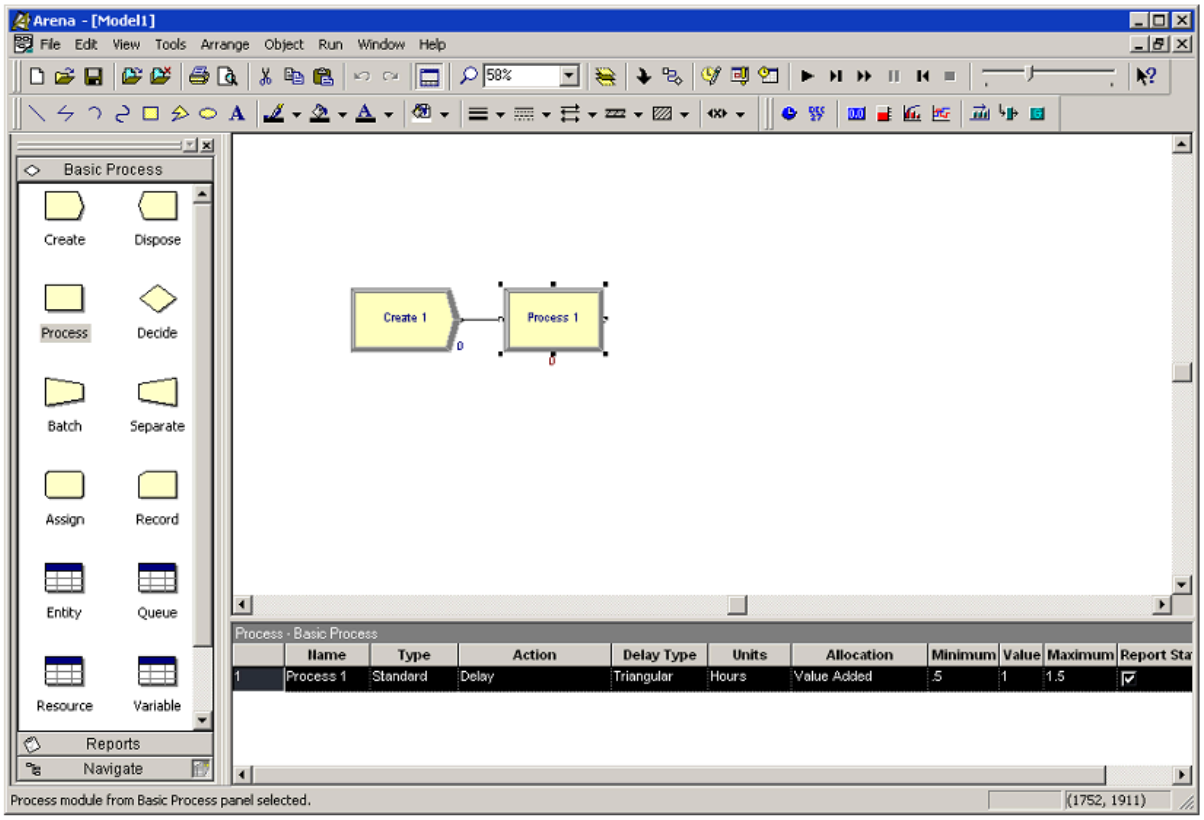
We'll start the flowchart using a Create module, from the Basic Process panel. This is the starting point for the flow of entities through the model.Drag the Create module from the Basic Process panel into the model window. A default name, Create 1, is given to the module when it's placed.

We'll return later to provide a more meaningful description as well as some data to support the simulation.



PROCESS THE IDENTIFICATION REVIEW

Next in our flowchart is a Process module, from the Basic Process panel, representing the Review Application step. Drag a Process module from the Basic Process panel into the model window, placing it to the right of the Create.Connect the two modules manually or check the Object >Auto-Connect menu to verify that it's checked. If it's not, select it to turn on this option.As with the Create, the Process module has a default name that we'll replace later.

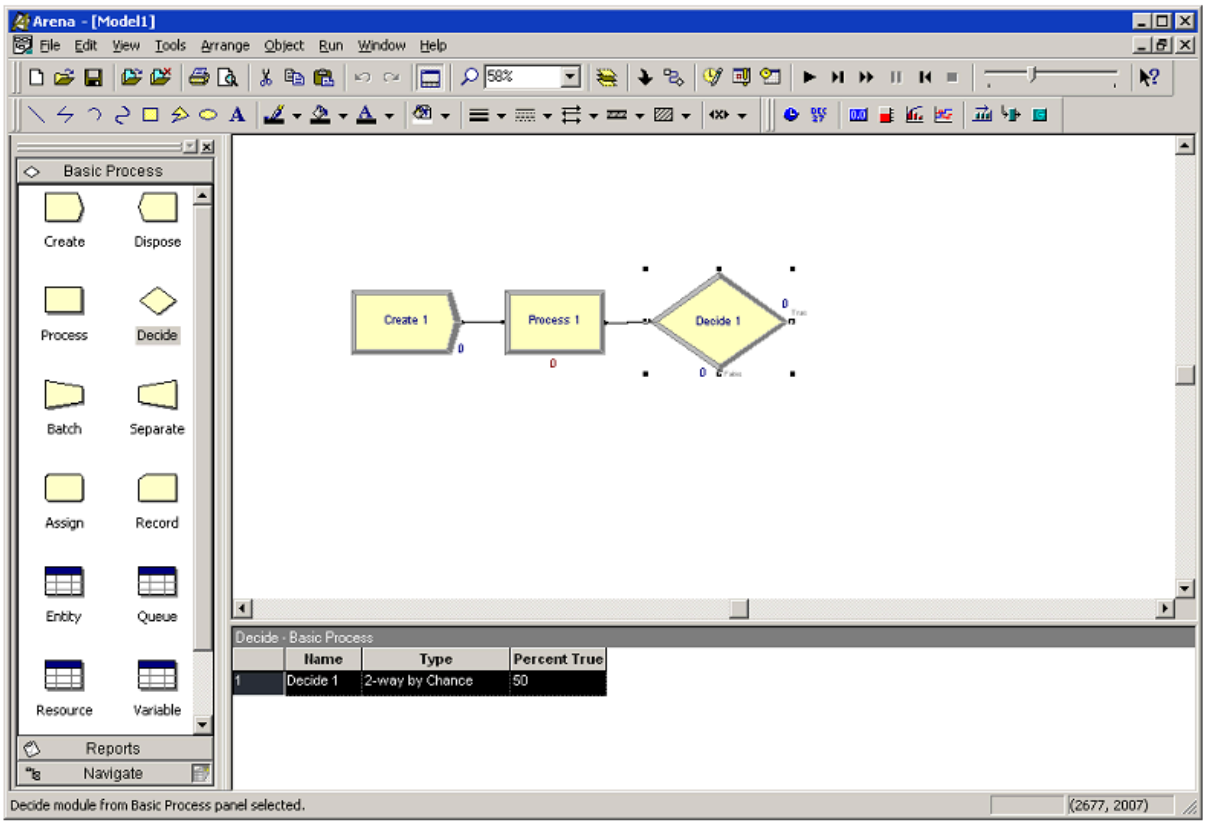


DECIDE WHETHER IDENTIFICATION REVIEW IS COMPLETE

After the Process, we have a Decide module, from the Basic Process panel, which determines whether the mortgage application is complete.

1. If you're using the Auto-Connect feature (i.e., it's checked on the Object > Auto-Connect menu), be sure that the Process module is selected so that the Decide will be connected to it.

2. Drag a Decide module to the right of the Process module.



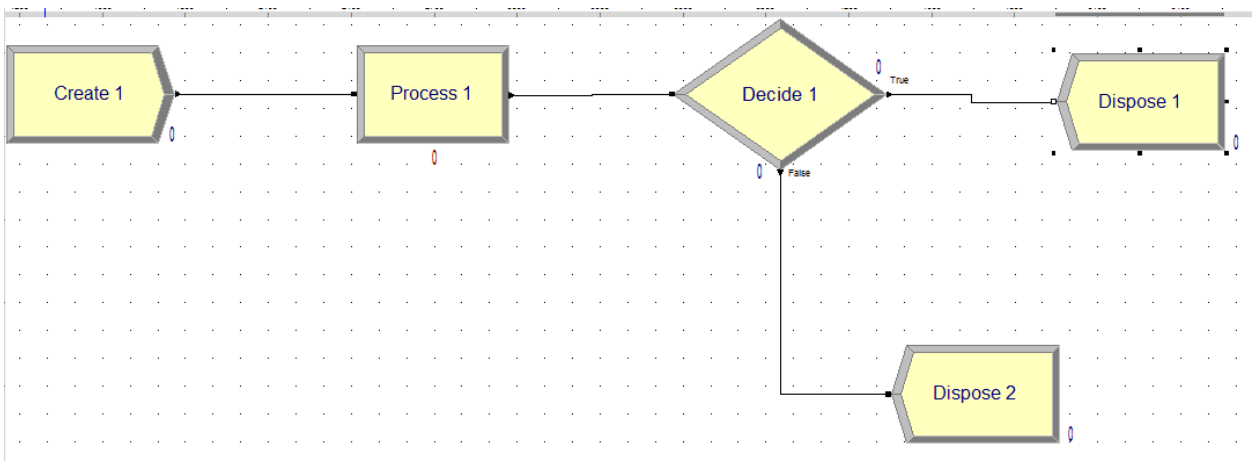
If the mortgage application has a complete set of information, it will leave the Decide module from the right side of the diamond shape, representing the True condition. Incomplete applications (False result to the Decide test) will leave via the bottom connection.

DISPOSE OF THE REVIEWS TO TERMINATE THE PROCESS

Next we'll place the Dispose module, from the Basic Process panel, representing accepted applications, connecting to the True (right) output from the Decide shape. Then, we'll complete the flowchart with another Dispose for returned applications.

1. Select the Decide shape so that our first Dispose will be connected automatically.
2. Drag a Dispose module to the right of the Decide module. Arena will connect it to the primary (True) exit point of the Decide module.
3. To add the second Dispose module, once again select the Decide module, so that Arena will automatically connect its False exit point to the new
4. Dispose module, and drag another Dispose module below and to the right of the Decide module.

Drag and drop another Dispose module, placing it below and to the right of the Decide shape, completing the process flowchart



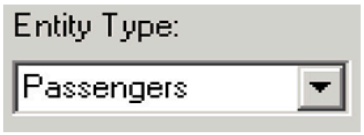
Entity flow always begins with a Create module and terminates with a Dispose module. You may have as many of each of these modules as you need to generate entities into the model and to remove them when their processing is complete.

INITIATE IDENTIFICATION REVIEW (CREATE MODULE)

First, let's rename the Create module to *Passengers Arrive to Security*. Its data will include the type of entity to be created—in our case, a passenger identification review. We also need to define how often identification reviews are initiated. We'll use an average of 2 minutes between passenger arrivals for review as a starting point for our model, and we'll make this a random activity to represent the natural variation in the timing of passenger reviews being submitted.

1. Double-click the Create module to open its property dialog box.
2. In the Name field, type **`Passengers Arrive to Security`**.

3. For the Entity Type, name our entities by typing **Passengers**.



4. Type **2** in the Value field of the Time Between Arrivals section.

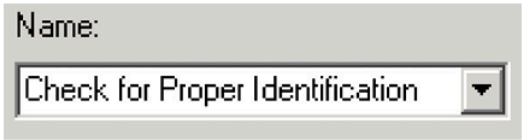
5. Click **OK** to close the dialog box.

REVIEW PASSENGER IDENTIFICATION (PROCESS MODULE)

Remember that as we create the flowchart, we're looking at the process from the perspective of the entity. The Create module is a starting point for an entity's flow through the system being modeled. Next, in our case, the passenger identification will be reviewed for completeness by a *Secutiry Agent*. Because this will take some amount of time, holding the entity at this point in the flowchart for a *delay* and requiring a *resource* to perform the activity, we use a Process module. We'll call this process *Check for Proper Identification*.

For our Check for Proper Identification process, we'll use a minimum time of .75 minute, most likely value of 1.5 minutes, and a maximum of 3 minutes. We will assign a *resource*, the *Transportation Security Officer*, to perform this process.

1. Double-click the Process module to open its property dialog box.
2. In the Name field, type **Check for Proper Identification**.



3. To define a resource to perform this process, pull down the **Action** list and select **Seize Delay Release**.

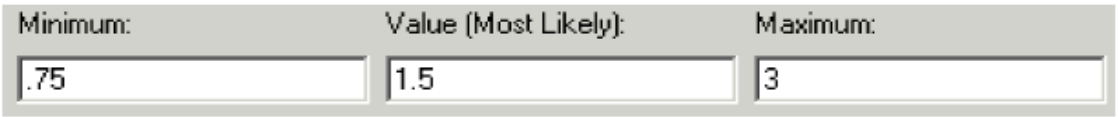
   Arriving entities will wait their turn for the resource to be available. When its turn comes, each entity will *seize* the resource, *delay* for the process time, and then *release* the resource to do other work.

4. A list of resources will appear in the center of the dialog box. To add a resource for this process, click **Add**.

If more than one resource is required for a process to be performed, add as many as are necessary in the Process dialog's Resources list. An entity won't commence its process delay until all listed resources are available.

5. In the Resource Name field of the Resource dialog box, type **`Transportation Security Officer`**.

6. Click **OK** to close the Resource dialog box.
7. Define the process delay parameters in the Minimum, Value (Most Likely), and Maximum fields as **`.75`**, **`1.5`**, and **`3`**. (Note that the default delay type is triangular and the default time units are in minutes.)



8. Click **OK** to close the dialog box.

   For now, we'll leave the default values for the other Process module properties. Feel free to explore their purposes through online help or the *Basic Concepts* and *Resources* models in the Smarts library.

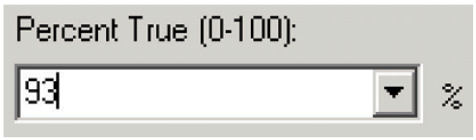
PASS SECURITY? (DECIDE MODULE)

After each passenger's identification has been reviewed, we determine whether to pass or fail the passenger through security. In Arena, whenever an entity selects among branches in the process logic, taking just one of the alternatives, a Decide module is used.

For the airport security review process, we'll use a simple probability to determine the outcome of the decision, with 93% of passenger reviews accepted as passed.

1. Double-click the Decide module to open its property dialog box.
2. In the Name field, type **`Pass Security?`**.

3. For the Percent True field, type **`93`** to define the percentage of entities that will be treated with a "True" decision (that is, will depart through the exit point at the right of the Decide module).



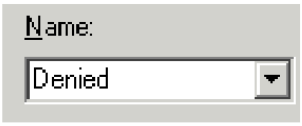
4. Click **OK** to close the dialog box.

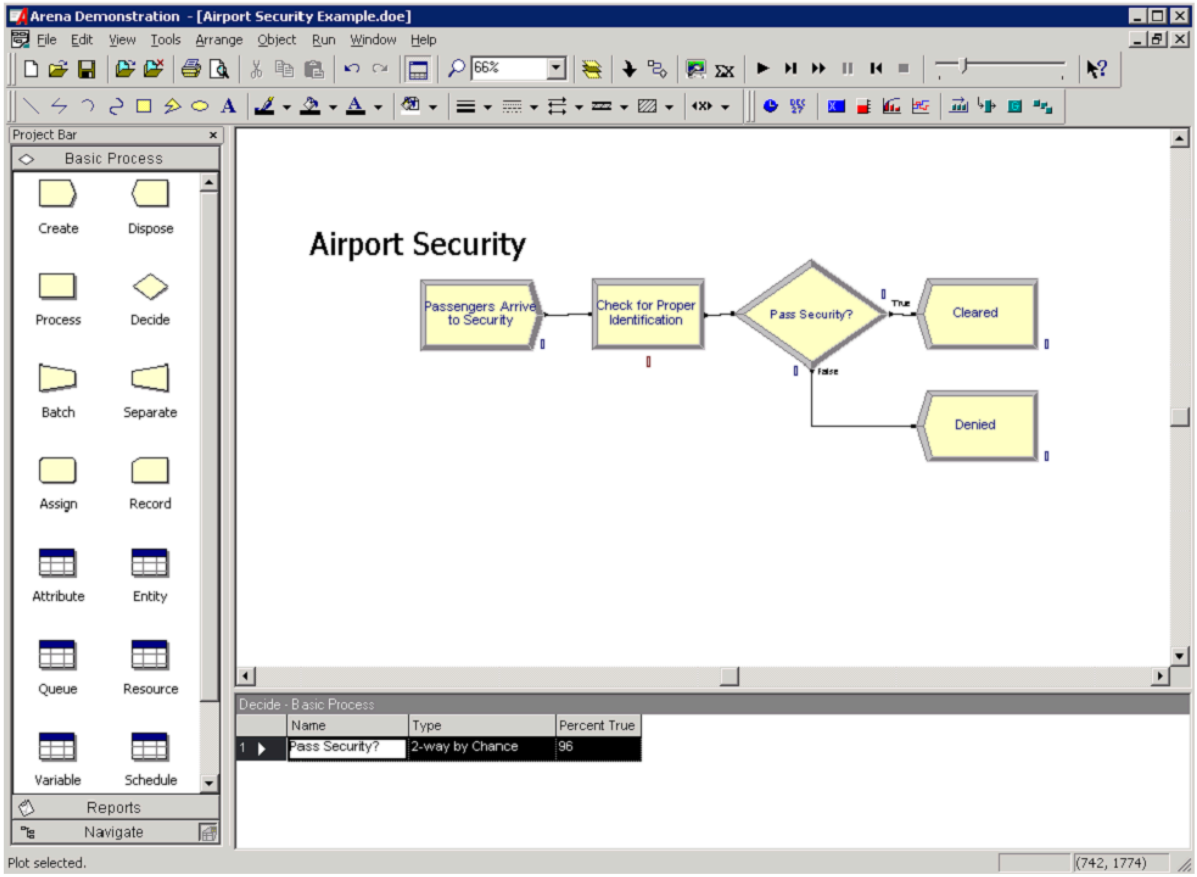
CLEARED, DENIED (DISPOSE MODULE)

In our simple process for reviewing passengers' identifications at airport security, all the work that we're interested in is done. Now, we'll remove the passengers from the model, terminating the process with a Dispose module. Because there are two possible outcomes of the security passenger identification process—passengers can be cleared or denied—we're using two Dispose modules that will count the number of applications under each outcome.

1. Double-click the first Dispose module (connected to the True condition branch of the Decide module) to open its property dialog box, and in the Name field, type **`Cleared`**.

2. Click **OK** to close the dialog box.
3. Double-click the other Dispose module to open its property dialog box. In the Name field, type **`Denied`**.





TRANSPORTATION SECURITY (RESOURCE MODULE)

Along with our flowchart, we also can define parameters associated with other elements of our model, such as resources, entities and queues. For the airport security process, we'll simply define the cost rate for the security officer so that our simulation results will report the cost associated with performing this process. The officer's costs are fixed at $12 per hour.

To provide these parameters to the Arena model, you'll enter them in the Resources spreadsheet.

1. In the Basic Process panel, click the Resource icon to display the Resources spreadsheet.

2. Because we defined the security officer as the resource in the Check for Proper Identification process, Arena has automatically added a resource with this name in the Resources spreadsheet. Click in the Busy/Hour cell and define the cost rate when the agent is busy by typing **12**. Click in the Idle/Hour cell and assign the idle cost rate by typing **12**.

Resource - Basic Process

	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures	Report Statistics
1 ▶	Transportation Security Officer	Fixed Capacity	1	12	12	0.0		0 rows	☑
	Double-click here to add a new row.								

PREPARE FOR THE SIMULATION

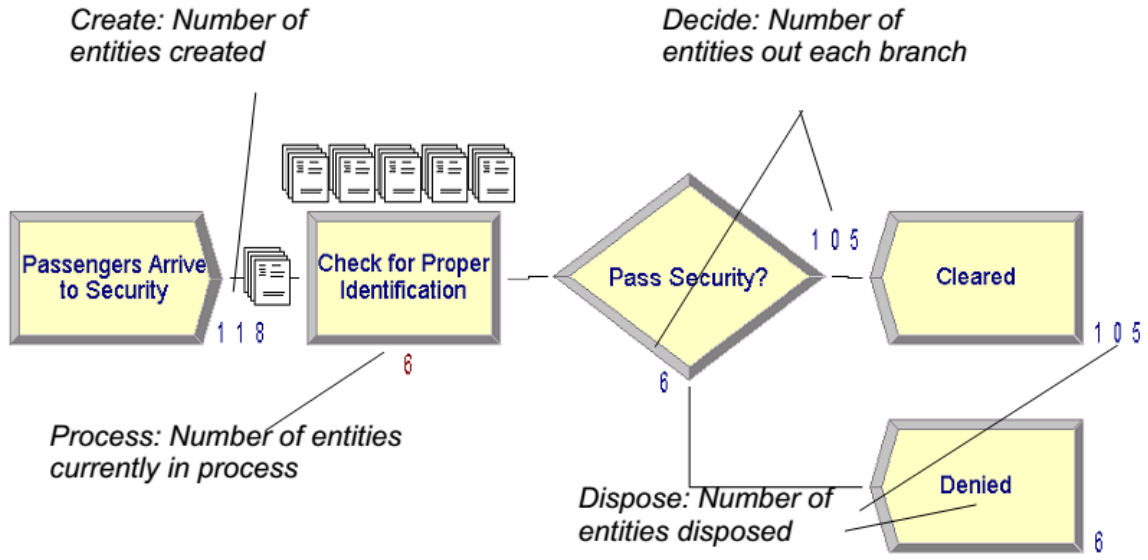
To make the model ready for simulation, we'll specify general project information and the duration of the simulation run. Since we're just testing our first-cut model, we'll perform a short, 20-day run.

1. Open the Project Parameters dialog box by using the **Run > Setup** menu item and clicking the **Project Parameters** tab. In the Project Title field, type **`Airport Security Analysis.`** We'll leave the Statistics Collection check boxes as the defaults, with Entities, Queues, Resources and Processes checked, and we'll also check the Costing box.
2. Next, click the **Replication Parameters** tab within the same Run Setup dialog box. In the Replication Length field, type **`20`**; and in the Time Units field directly to the right of Replication Length, select **days** from the drop-down list. Click **OK** to close the dialog box.

SIMULATE THE PROCESS

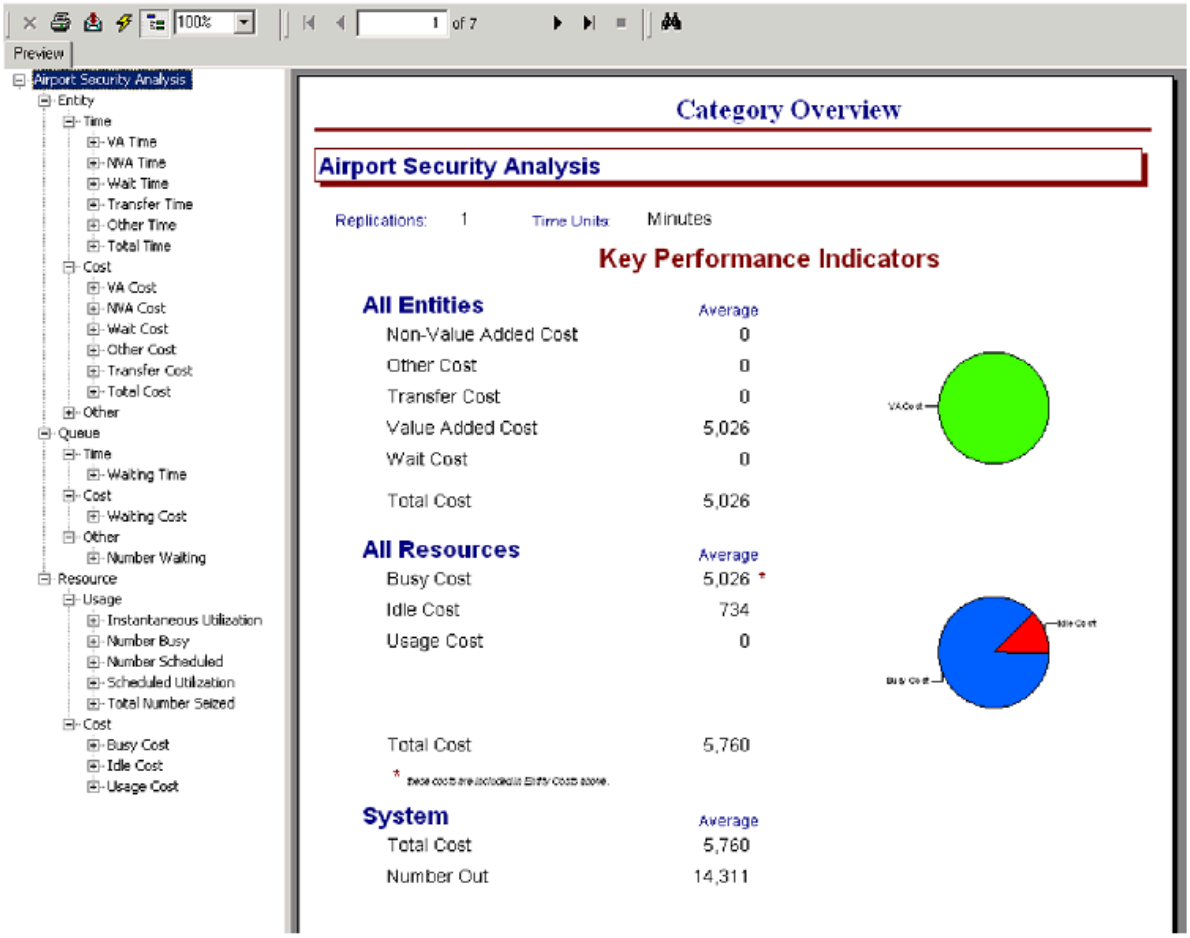
With these few, short steps, we are ready to predict the future! The mortgage application model contains all of the information needed to run the simulation. Start the simulation run by clicking the Go button or clicking the Run > Go menu item. Arena first will check to determine whether you've defined a valid model, then will launch the simulation.

As the simulation progresses, you'll see small entity pictures resembling report documents moving among the flowchart shapes. Also, a variety of variables change values as entities are created and processed, as illustrated below.



View Simulation Reports

After you've watched some of the animated flowchart, you can quickly run to the end of the simulation to view reports. Pause the simulation, then click the Fast Forward button to run the simulation without updating the animation. At the end of the run, Arena will ask whether you'd like to view reports. Click Yes, and the default report (the Category Overview Report) will be displayed in a report window, as shown below.



Comments

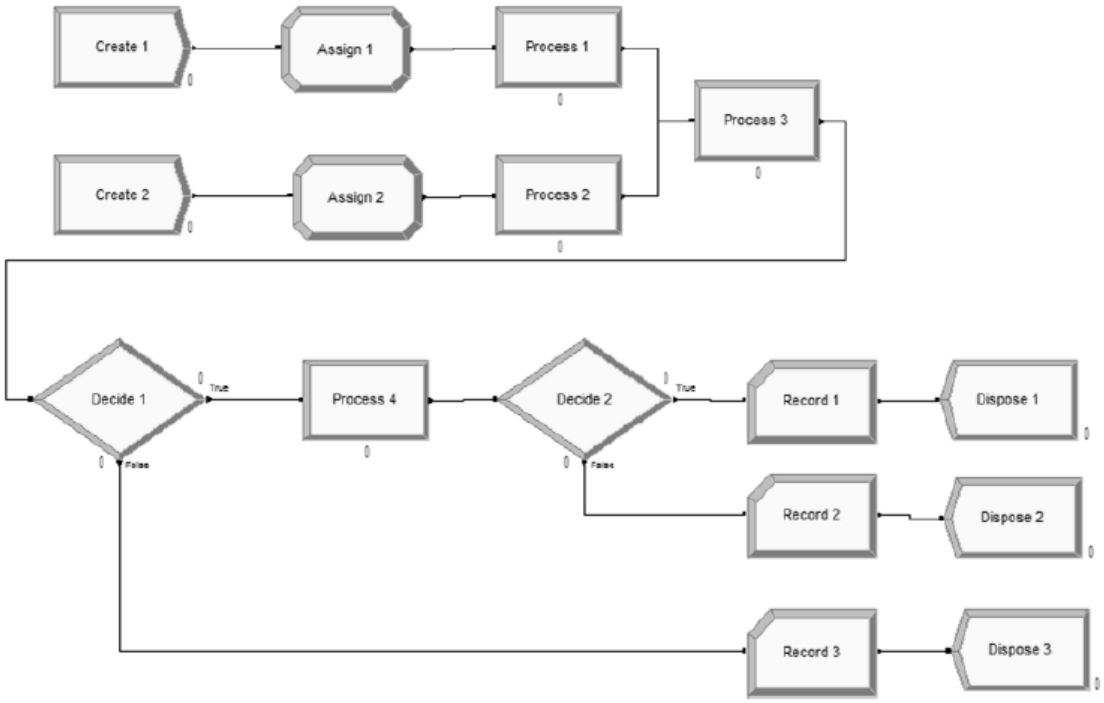
Lab Session No.8

Simulate and analyze a small manufacturing system with two processes in parallel.

Theory

OUR TASK

- Produce two different sealed elect. units (A, B)
- Arriving parts: cast metal cases machined to accept the electronic parts
- Part A, Part B – separate prep areas
- Both go to Sealer for assembly, testing – then to Shipping (out) if OK, or else to Rework
- Rework – Salvaged (and Shipped), or Scrapped



Part A

- Interarrivals: expo (5) minutes
- From arrival point, proceed immediately to Part A Prep area
    - Process = (machine + deburr + clean) ~ tria (1,4,8) minutes

- Go immediately to Sealer
    - Process = (assemble + test) ~ tria (1,3,4) min.
    - 91% pass, go to Shipped; Else go to Rework
- Rework: (re-process + testing) ~ expo (45)
    - 80% pass, go to Salvaged; Else go to Scrapped

Part B

- Interarrivals: *batches* of 4, expo (30) min.
- Upon arrival, batch separates into 4 individual parts
- From arrival point, proceed immediately to Part B Prep area
    - Process = (machine + deburr +clean) ~ tria (3,5,10)
- Go to Sealer
    - Process = (assemble + test) ~ weib (2.5, 5.3) min. , *different* from Part A, though at same station
    - 91% pass, go to Shipped; Else go to Rework
- Rework: (re-process + test) = expo (45) min.
    - 80% pass, go to Salvaged; Else go to Scrapped

Part A Create Module

- Name: Part A Arrive
- Entity Type: Part A
- Time Between Arrivals
    - Type: Random (Expo)
        - Pull-down list with options

        - Value: 5
        - Units: Minutes
            - Pull-down list with options
- Default what's not mentioned above

Part B Create Module

- Name: Part B Arrive
- Entity Type: Part B
- Time Between Arrivals
    - Type: Random (Expo)
        - Pull-down list with options
    - Value: 30
    - Units: Minutes
        - Pull-down list with options
- Entities per Arrival: 4

Part A Attributes Assign Module

- Name: Assign Part A Sealer and Arrive Time
- Add button:
    - Type: Attribute
    - Attribute Name: Sealer Time
    - New Value: TRIA(1, 3, 4)
- Add button:
    - Type: Attribute

        - Attribute Name: Arrive Time
        - New Value: TNOW (to compute time in system on exit)

Part B Attributes Assign Module

- Name: Assign Part B Sealer and Arrive Time
- Add button:
    - Type: Attribute
    - Attribute Name: Sealer Time
    - New Value: WEIB(2.5, 5.3)
- Add button:
    - Type: Attribute
    - Attribute Name: Arrive Time
    - New Value: TNOW

Prep A Process Module

- Name: Prep A Process
- Action: Seize Delay Release
- Resources subdialog (Add button):
    - Type: Resource (a pull-down option)
    - Resource Name: Prep A
    - Quantity: 1 (default)
- Delay Type: Triangular
- Units: Minutes

- Minimum: 1
- Value (Most Likely): 4
- Maximum: 8

Prep B Process Module

- Name: Prep B Process
- Action: Seize Delay Release
- Resources subdialog (Add button):
    - Type: Resource (a pull-down option)
    - Resource Name: Prep B
    - Quantity: 1 (default)
- Delay Type: Triangular
- Units: Minutes
- Minimum: 3
- Value (Most Likely): 5
- Maximum: 10

Sealer Process Module

- Name: Sealer Process
- Action: Seize Delay Release
- Resources subdialog (Add button):
    - Type: Resource (a pull-down option)
    - Resource Name: Sealer
    - Quantity: 1 (default)
- Delay Type: Expression
- Units: Minutes

- Expression: Sealer Time

Sealer Inspection-Result *Decide* Module

- Decide module provides branch points
    - *By Condition* (entity Attributes, global Variables)
    - *By Chance* (multi-sided, possibly-biased hypercoin flip)
- Name: Failed Sealer Inspection
- Type: 2-way by Chance (default)
- Percent True: 9
- Different exit points for True, False results – connect appropriately downstream

Rework Process Module

- Name: Rework Process
- Action: Seize Delay Release
- Resources subdialog (Add button):
    - Type: Resource (a pull-down option)
    - Resource Name: Rework
    - Quantity: 1 (default)
- Delay Type: Expression
- Units: Minutes
- Expression: EXPO(45)

Rework Inspection-Result Decide Module

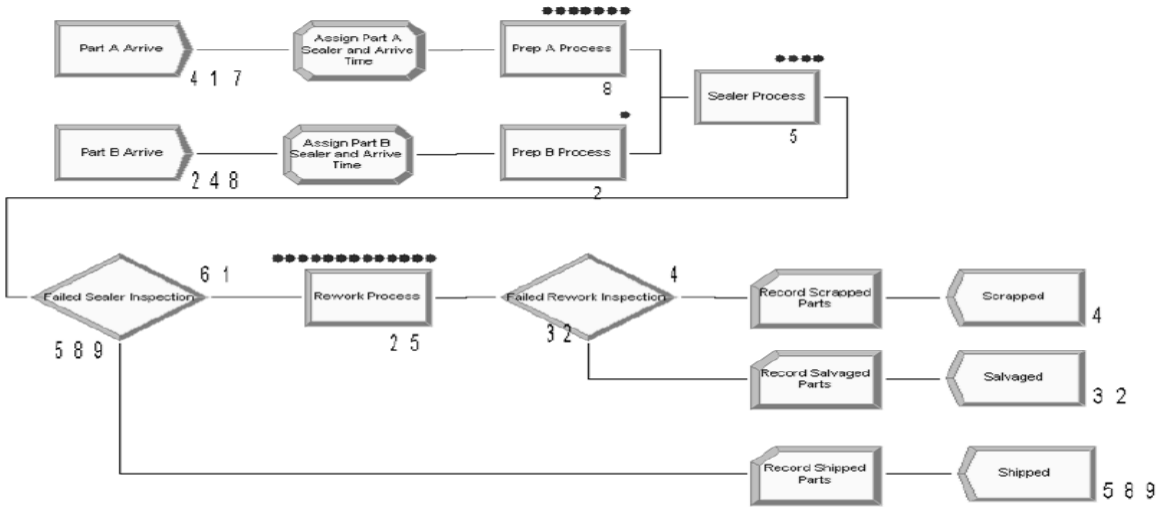
- Name: Failed Rework Inspection
- Type: 2-way by Chance (default)
- Percent True: 20

Record Modules

- Arena collects and reports many output statistics by default, but sometimes not all you want
- We want time in system (average, max) of parts sorted out by their exit point (Shipped, Salvaged, Scrapped)
    - It's this sorting that Arena doesn't do by default ... it would automatically sort *by Entity Type* if we had Entities checked in Run > Setup > Project Parameters (which we don't)
- Record module can be placed in the flowchart to collect and report various kinds of statistics from within the model run as entities pass through it

Shipped Parts *Record* Module

- Name: Record Shipped Parts
- Type: Time Interval
    - This option records the length of time that elapsed up to now (TNOW) from when an entity attribute was marked with a time "stamp" upstream ... Attribute Name is below ...
    - There are several other options for Type … explore via Record module's Help button!
- Attribute Name: Arrive Time
    - Was defined upstream as the clock value in the Assign modules instantly after each entity was Created
- Tally Name: Record Shipped Parts
    - Determines the label in the reports



Dispose Modules

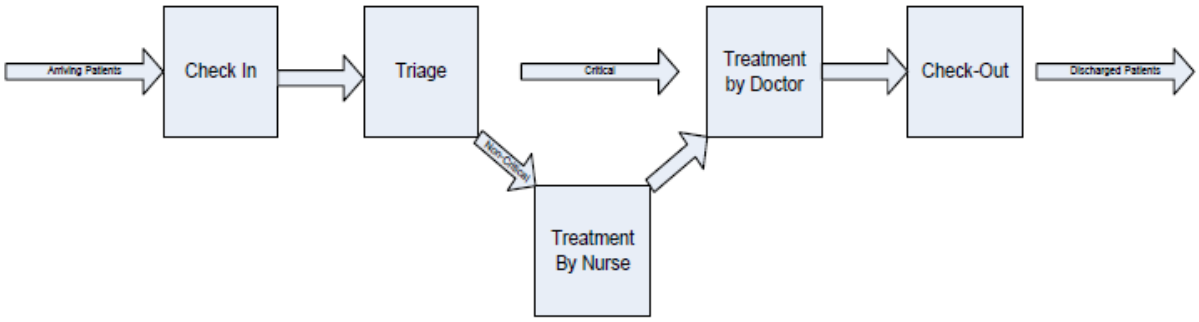
- Three separate exit points for three separate part disposition (Shipped, Salvaged, Scrapped)
- Could have directed all three exit types to a single Dispose module
    - But having separate ones produces animation counts of the three dispositions
- Also, having separate Dispose modules allows for differentially checking the boxes to Record Entity Statistics
    - Produces flow statistics separated by entity type (*if* Entities Statistics Collection is checked in *Run* > *Setup* > *Project Parameters*), *not* by final disposition of part … so we *did* need our Record modules and Arrive Time attribute

Lab Session No.9

To build simulation model of a hospital emergency room.

Theory

The emergency room of a small hospital operates around the clock. It is staffed by three receptionists at the reception office, and two doctors on the premises, assisted by two nurses. However, one additional doctor is on call at all times; this doctor is summoned when the patient workload up-crosses some threshold, and is dismissed when the number of patients to be examined goes down to zero, possibly to be summoned again later. Figure 3.1 depicts a diagram of patient sojourn in the emergency room system, from arrival to discharge.



Patients arrive at the emergency room according to a Poisson process with mean interarrival time of 10 minutes. An incoming patient is first checked into the emergency room by a receptionist at the reception office. Check-in time is uniform between 6 and 12 minutes. Since critically ill patients get treatment priority over noncritical ones, each patient firstundergoes triage in the sense that a doctor determines the criticality level of the incoming patient in FIFO order. The triage time distribution is triangular with a minimum of 3 minutes, a maximum of 15 minutes, and a most likely value of 5 minutes. It has been observed that 40% of incoming patients arrive in critical condition, and such patients proceed directly to an adjacent treatment room, where they wait FIFO to be treated by a doctor. The treatment time of critical patients is uniform between 20 and 30 minutes. In contrast, patients deemed noncritical first wait to be called by a nurse who walks them to a treatment room some distance away. The time spent to reach the treatment room is uniform between 1 and 3 minutes and the treatment time by a nurse is uniform between 3 and 10 minutes. Once treated by a nurse, a noncritical patient waits FIFO for a doctor to approve the treatment, which takes a uniform time between 5 to 10 minutes. Recall that the queueing discipline of all patients awaiting doctor treatment is FIFO within their priority classes, that is, all patients wait FIFO for an available doctor, but critical patients are given priority over noncritical ones. Following treatment by a doctor, all patients are checked out FIFO at the reception office, which takes a uniform time between 10 and 20 minutes, following which the patients leave the emergency room.
The performance metrics of interest in this problem are as follows:

- Utilization of the emergency room staff by type (doctors, nurses, and receptionists)
- Distribution of the number of doctors present in the emergency room
- Average waiting time of incoming patients for triage
- Average patient sojourn time in the emergency room
- Average daily throughput (patients treated per day) of the emergency room

Arena Model

We now proceed to construct an Arena model of the system under study. Figure depicts an Arena model of emergency room system, where modules are labeled with their type to facilitate understanding.

The model is composed of two segments:

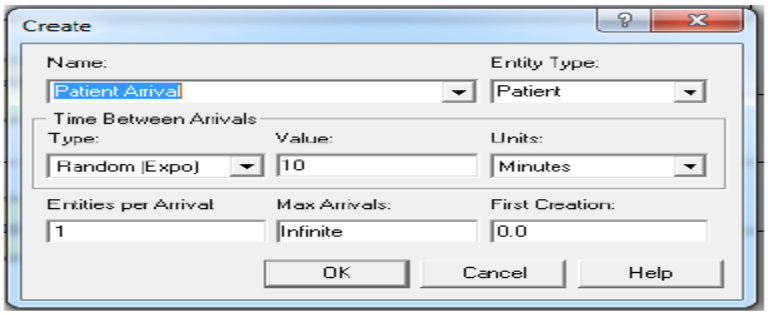
- ***Emergency room segment.*** This logic (top segment of Figure 5.21) keeps track of model entities (patients in our case), whose specification can be viewed and edited in the spreadsheet view of the Entity module from the Basic Process template panel. In this part of the model logic, a patient is generated and then moves through the emergency room processing, from admission to treatment to discharge.
- ***On-call doctor segment.*** This logic (bottom segment of Figure 5.21) controls the periodic summoning and dismissal of the extra doctor on call. This is achieved by a perpetually circulating single entity, called administrator in this model, which triggers the summoning and dismissal of the on-call doctor.

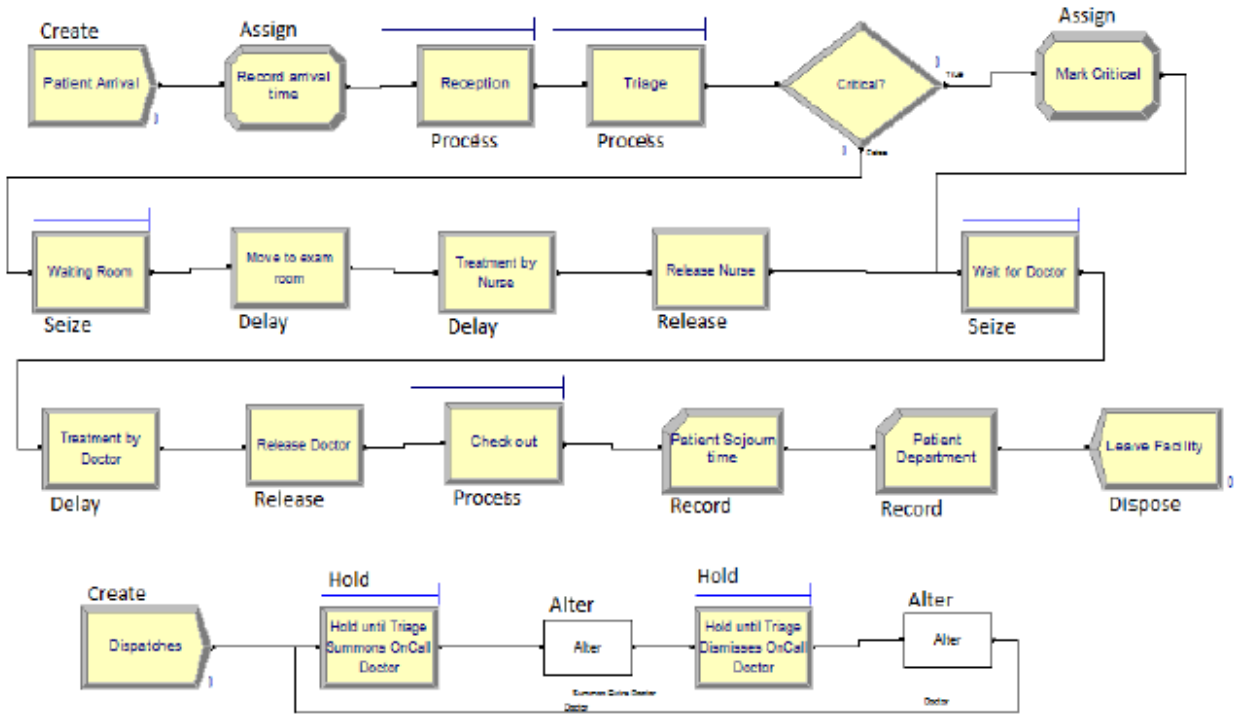
In addition, input and output data logic is interspersed in the two segments above. This logic consists of input/output modules (corresponding to variables, resources, statistics, etc.) that set input variables, compute statistics, and generate summary reports. We assume that the emergency room is initially empty and the on-call doctor is not present.

Emergency Room Segment

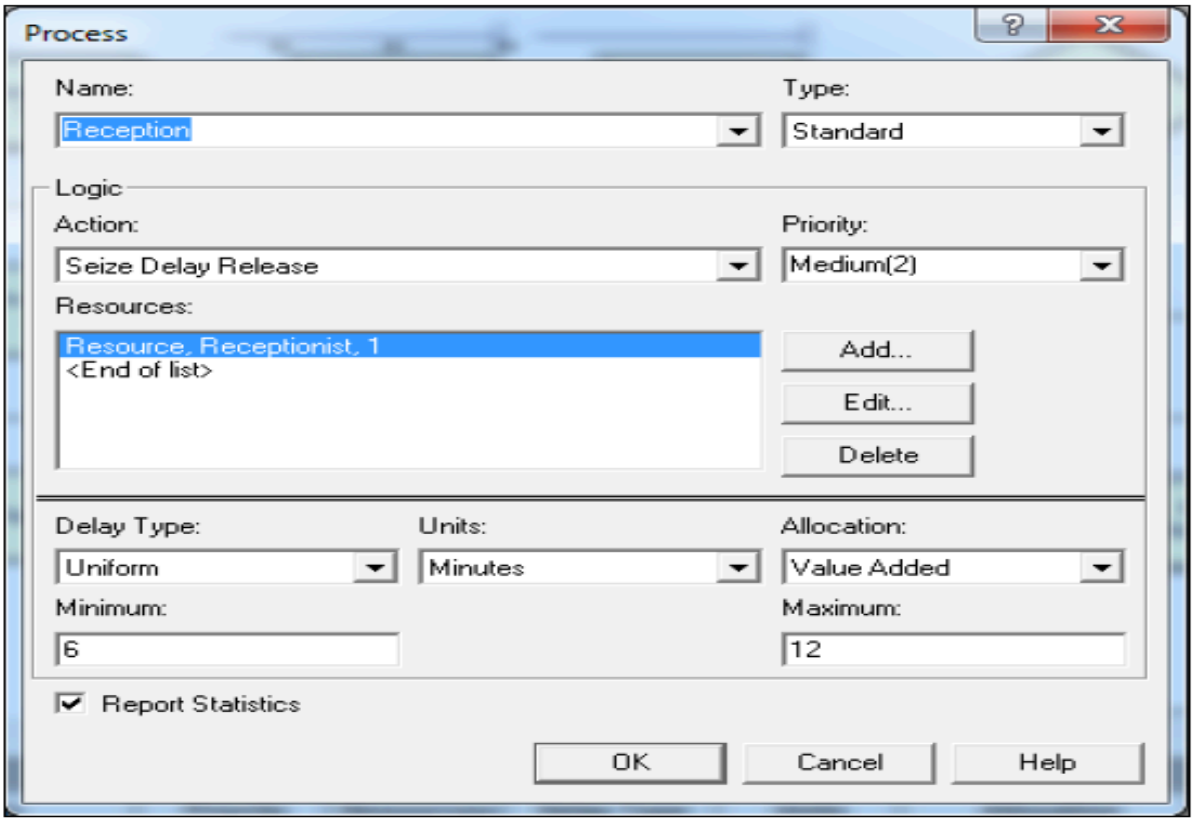
Starting at top left and moving to the right in the emergency room segment, the first module is the Create module, called Patient Arrivals, which generates incoming patient entities. Figure 3.3 displays the dialog box for this module, showing that patients arrive according to a Poisson process with exponential interarrival times of mean 10 minutes.

An incoming patient entity then enters the Assign module, called Record Arrival Time, where its arrival time is recorded in its ArrTime attribute. The value in this attribute will be carried by the patient entity throughout its sojourn in the emergency room system, and will be used later to compute its sojourn time (total time in the system).



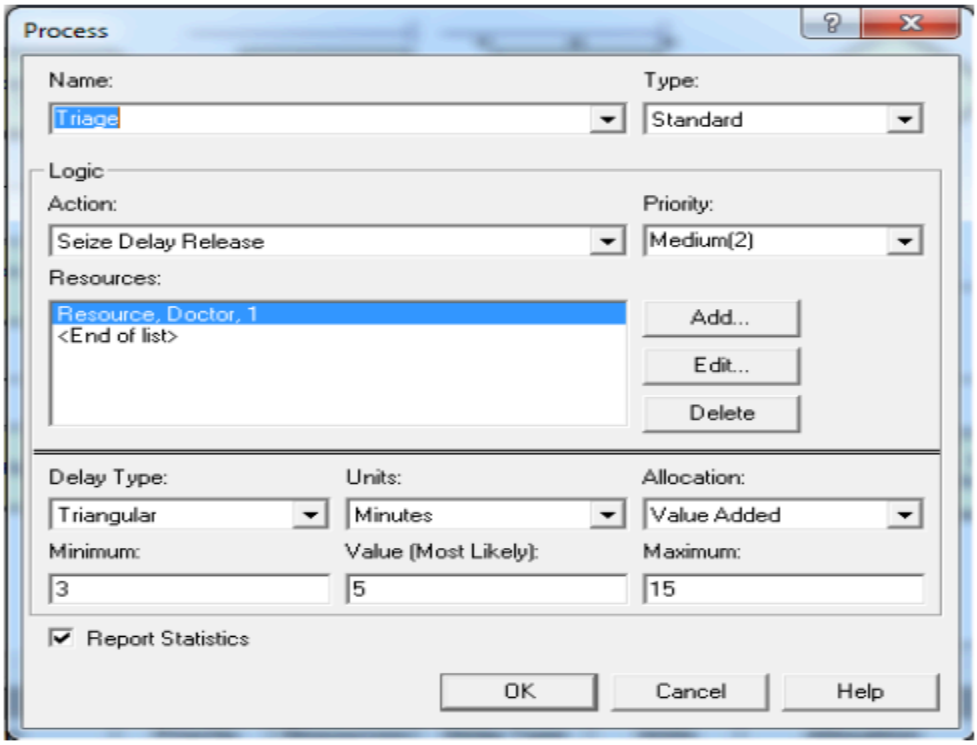


ARENA MODEL FOR EMERGENCY ROOM

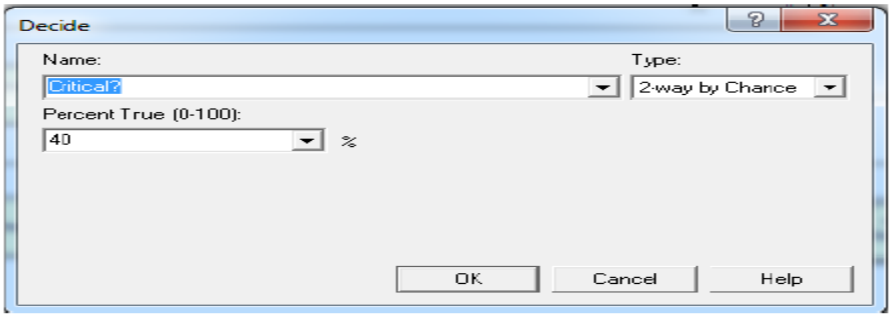


The patient entity then promptly attempts to check into the emergency room by entering the Process module, called Reception, whose dialog box is depicted in Figure 4.3

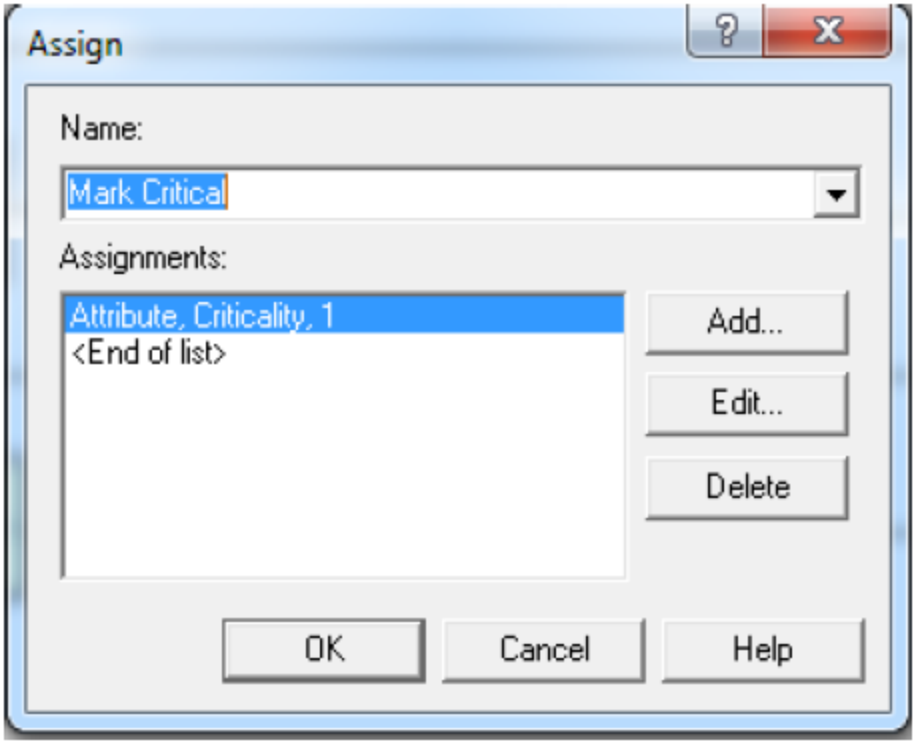
At this juncture, the patient entity waits in line (if any) to seize a receptionist for a uniform check-in processing time between 6 and 12 minutes, after which the receptionist is released and becomes available to other patient entities. Note that the Process module uses the Seize Delay Release option in the Action field, since receptionists are modeled as a resource, and the problem calls for computing expected waiting times and utilizations of the check-in operation. Once check-in is completed, the patient entity proceeds to the Process module, called Triage, to undergo a triage checkout by a doctor. Figure 3.5 shows that a patient entity waits for a doctor to become available, and then undergoes a random triage time, drawn from the triangular distribution between 3 and 15 minutes, with a most likely time of 5 minutes.



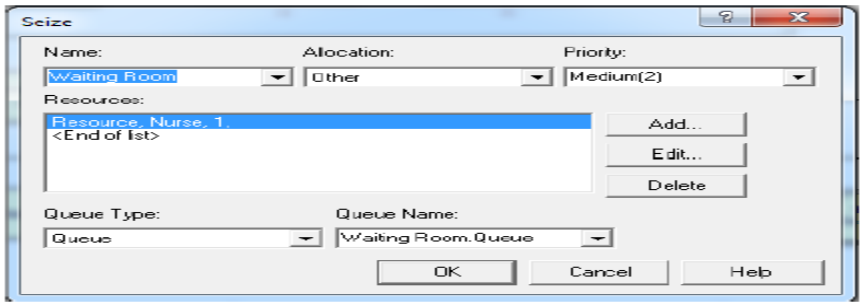
After the triage delay is completed, the triage doctor is released and the patient entity proceeds to determine its level of criticality. To this end, it enters the Decide module, called Critical?, whose dialog box is depicted in Figure



- The true branch indicates that the patient is entity deemed critical. Such a patient entity will proceed to be treated by a doctor. Recall that this applies to 40% of the patients.
- The false branch indicates that the patient entity is deemed noncritical. Such a patient entity will proceed to be treated by a nurse, and then will be inspected by a doctor, but at a lower priority than any critical patient entity.



The criticality level of patient entities is indicated in their Criticality attribute: a value of 1 codes for a critical patient, while a value of 0 codes for a noncritical patient. Accordingly, critical patient entities exiting module Critical? proceed to the Assign module, called Mark Critical, where their Criticality attribute is set to 1, as shown in the dialog box of Figure 4.7
In contrast, noncritical patient entities are automatically marked as such, since the default value of the Criticality attribute is 0 (recall that this is the Arena convention for all attributes). Such patient entities exiting module Critical? proceed to the Seize module, called Waiting Room, whose dialog box is depicted in Figure.



In this module, noncritical patient entities wait FIFO in a queue, called Waiting Room. Queue, for a nurse until one becomes available. Once a nurse is seized, the patient entity passes through two Delay modules in succession: module Move to Treatment Room models the uniformly distributed time between 1 and 3 minutes that it takes the nurse to walk a (noncritical) patient to a treatment room, while module Treatment by Nurse models the uniformly distributed time between 3 and 10 minutes that it takes the nurse to treat a patient. Figure 3.9 depicts the dialog

boxes of these two Delay modules. Having completed its treatment, the noncritical patient entity releases the nurse by entering the Release module, called Release Nurse, whose dialog box is shown in Figure 3.10. At this point the paths of critical and noncritical patient entities converge, and all patient entities, both critical and noncritical, attempt to enter the Seize module, called Wait for Doctor. Note that an individual Seize module has the same functionality as the Seize option in a Process module, but with the added flexibility that the modeler can insert extra logic between the Seize and Delay functionalities(this is impossible in a Process module).

Lab Session No.10

Construct simulation model for some food restaurant.

Theory

Entities:

• The dynamic objects in the simulation that move around, change status, affect and are affected by other entities and the state of the system, and affect the output performance.
• They usually are created, move around for a while and then are disposed (leave) E.g. parts to be processed, customers in a banking system, etc.

Resources:

• Entities often take the service from resources
• An entity seizes a resource when available and releases it. E.g. Machines, Server

Attributes:

• Attributes are generally attached to individual entities E.g. Part entities have attributes called due date, priority, color, etc

Variables:

- A piece of information that reflects some characteristic of your system, regardless of how many or what kinds of entities might be around.
- Many different variables are possible in a model.
- In Arena there are two types of variables: Built-in variables (number-in queue, number of busy servers, current simulation clock time, and so on).
- User-defined variables (mean service time, travel time, current shift, and so on).

Record Module:

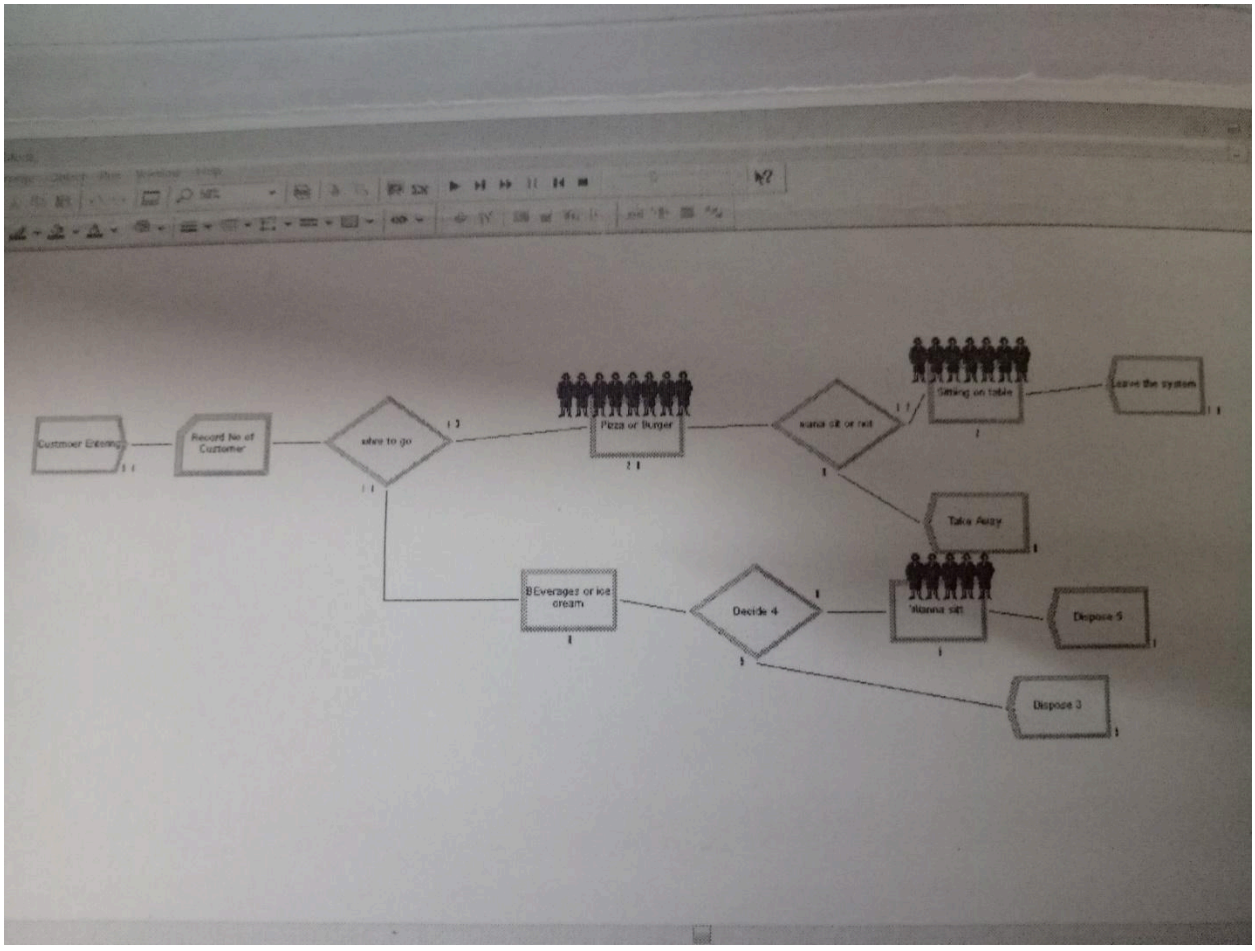
- The record module is used by entities to collect statisticsat selected locations in the model.

Decide Module:

- The decide module is used by entities to make branching decisions, based on chance or the truth/falsity of prescribed condition.

Procedure:

At first use the create module with name customer entering in restaurant. Time between arrivals are five minutes and entity per arrival is one. Then use record module to record number of customer entering in the restaurant. After that, use decide module to choose whether customer is going for pizza, burger or for ice cream, soft drink.If customer is going for ice cream or soft drink then again use decide module to choose that does customer want to sit or go. If go, we dispose directly. If sitting, use process module then dispose module. Same for the case of pizza and burger.



Lab Session No.11

To provide plan for car maintenance through simulation.

Theory

Endogenous:

- Activities and events occurring with the system.

Exogenous:

- Activities and events occurring with the environment.

Verification:

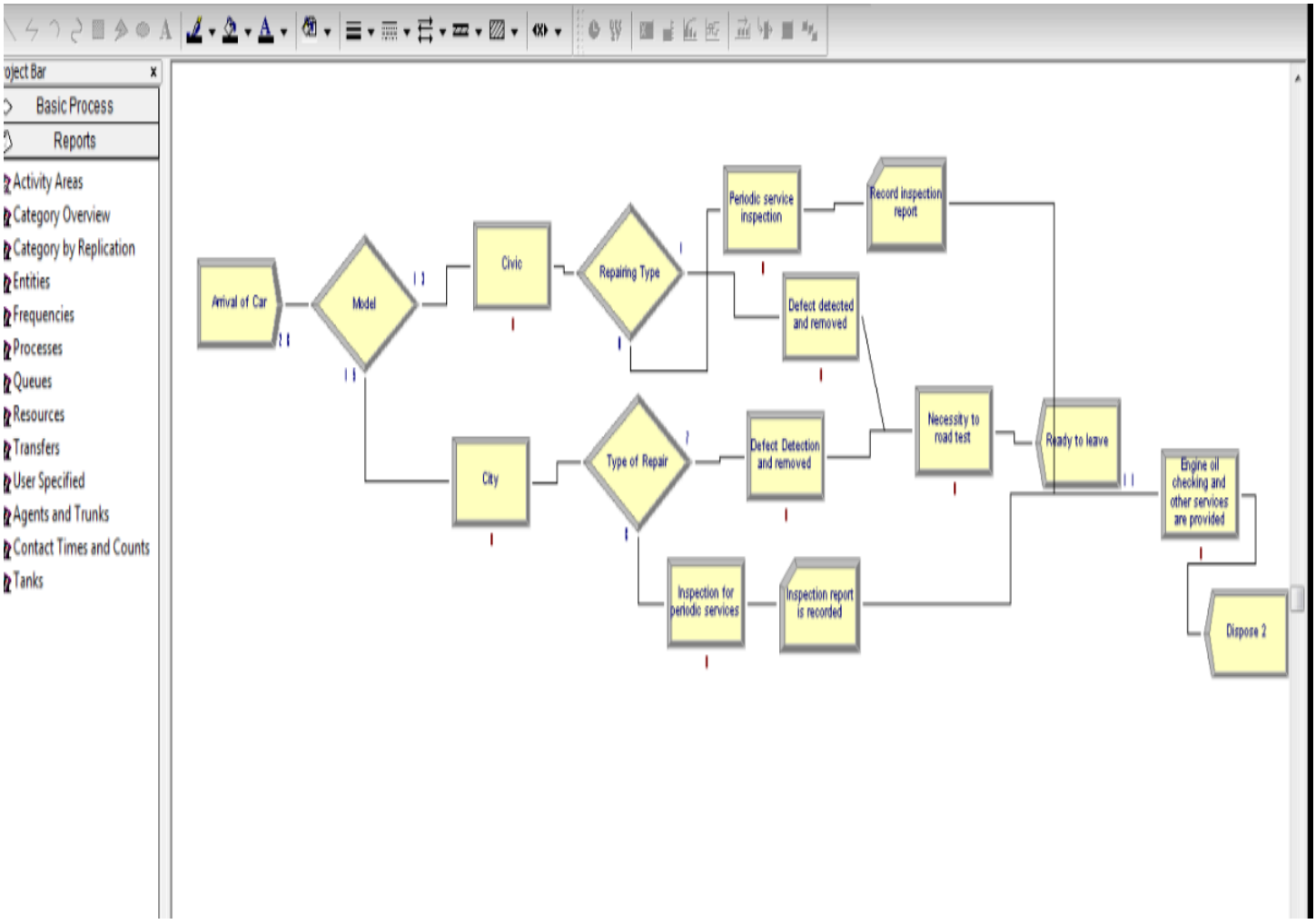
- Computational model should be consistent with specification model
- Did we build the model right?

Validation:

- Computational model should be consistent with the system being analyzed.
- Did we build the right model?
- Can an expert distinguish simulation output from system output?

Procedure:

Use create module for arrival or car. User decision module to decide model of car. One model is civic and one is city. Again use decision module for both models separately to decide type or repair. Use two process module and connect with previous decision module. One for periodic service inspection and one for defect detection and removal for both. For inspection and services use recorder module to record inspection report and for defect use process module to test the defect and if there is no defect part is ready to leave system so use dispose module. Use process module for both models to check engine oil and provide other services and use dispose to dispose the system.



Lab Session No.12

CS312L_CS_V1_F2018

To simulate a coin toss in spreadsheet.

Theory

How to Use the Spreadsheets

The spreadsheet simulation models are in the 'One Trial'worksheet of each solution workbook. This sheet contains the definition and specification of model inputs, the simulation table containing the formulas to generate each row to carry the simulation forward, and some summary statistics (and usually a histogram) representing the results of one simulation run. The two buttons on the worksheet control the simulation:

1. On each click, the 'Generate New Trial' button makes a new run of the simulation, using a new sequence of random numbers each time.
2. . On each click, the 'Reset Seed & Run' button sets the RNG seed to the value specified by the user in an adjacent cell, and then runs the simulation using this seed. The default seed isthe arbitrary value 12,345.

You can always return to the initial results by clicking the 'Reset Seed & Run' button.
The 'Experiment' worksheet allows a user to conduct a simulation experiment by repeatedly executing the 'One Trial' worksheet, each time with different random numbers, and recording the designated model response (from the Link cell) to the Response Table. The user specifies the number of trials, for example, 100, meaning that the 'One Trial' worksheet will be executed 100 times and the 100 response values recorded in the Response Table. This is called replication; the number of replications is the number of trials.
The experimental results over all trials are summarized in the "Multi-Trial Summary" table on the 'Experiment' sheet. These summary statistics include frequency distributions and an accompanying histogram, and standard statistics such as sample average, median, minimum, and maximum.

To run the experiment on the 'Experiment'worksheet, use the two buttons as follows:

1. On each click, the 'Run Experiment'button makes one run of the experiment for the specified number of trials. Each subsequent click runs the experiment using different random numbers.
2. On each click, the 'Reset Seed & Run' button sets the RNG seed to the specified value in an adjacent cell (default 12,345) and makes one run of the experiment. Each click reproduces the exact same original results.

The results given in all the examples in this chapter come from clicking the 'Reset Seed & Run' button; this guarantees that a reader can reproduce these same results. Each spreadsheet solution

contains comments and explanations on different aspects of the solution, including the implementation of model logic; some solutions have a worksheet named 'Explain' with lengthier explanations of certain aspects of the model.

Simulate a Coin Toss

How can we simulate a coin toss in a spreadsheet model? The solution shows how to use a uniform random number generator in a spreadsheet simulation.

We want to simulate a sequence of 10 coin tosses. The coin is a "fair" coin, meaning that the chance of getting a head is the same as getting a tail. Each has probability 0.5 of occurring. You should run the simulation multiple times and compare the results with your expectations from real-life coin tossing. This is an example of a Monte Carlo simulation; there are no events or clock times being tracked. The solution, shown in Table 1, is in the spreadsheet "Example2.1CoinToss.xls". In fact, there are four solutions in the spreadsheet, the first two using the RAND() random number generator, and the last two using the supplied VBA function Rnd01(). It is left as an exercise (at the end of the chapter) to devise a justification for the statistical equivalence of the 4 solutions, even though on any click of the button 'Generate New Trial', the 4 sets of results are different. The spreadsheet logic for coin tossing is simple.
The second solution (Column E) is:
=IF(RAND()<=0.5,"H","T")
while the fourth solution (Column I) is:
=IF(Rnd01()<=0.5,"H","T")

Each call to RAND() or Rnd01() produces a new random number. The 'One Trial' worksheet records the frequency for heads versus tails in the 10 tosses and graphs the results in a histogram.

Table 1 Simulation Table for Coin Tossing

	B	C	D	E	F	G	H	I
11		Solution #1A		Solution #1B		Solution #2A		Solution #2B
12								
13								
14	Toss	RAND()	Solution using RN from ColC	Solution using RAND()	Toss	Rnd01()	Solution using RN from ColG	Solution using Rnd01()
15	1	0.7317089	T	T	1	0.9871114	T	T
16	2	0.8285837	T	T	2	0.0225598	H	H
17	3	0.0701168	H	T	3	0.0008356	H	T
18	4	0.2147295	H	H	4	0.2127686	H	T
19	5	0.8613179	T	H	5	0.8586159	T	H
20	6	0.4159098	H	H	6	0.5503362	T	T
21	7	0.7492317	T	H	7	0.5243730	T	H
22	8	0.9671661	T	T	8	0.8884351	T	H
23	9	0.4823089	H	T	9	0.2111118	H	T
24	10	0.6738897	T	H	10	0.7680427	T	H

The first and third solutions are two steps, an approach that is useful when a generated random number is used more than once. Columns C and G contain the generated random number, and columns D and H the result (“H” or “T. For all future spreadsheet simulations, we use the supplied VBA function, Rnd01(), or one of theVBAfunctions based on Rnd01(), instead of RAND(). However, for serious professionalwork, we recommend using the generator implemented in the VBA function, MRG32k3a, from the file “RandomNumberTools.xls”.As previously mentioned, both the Excel worksheet function, RAND(), and the VBA function, Rnd(), have serious limitations and deficiencies, and should only be used for casual or educational purposes.

Lab Session No.13

To Simulate a random service time by using spreadsheet.

Theory

In this section our goal is to learn how to generate random service times and compute arrival times of customers or other entities to a system.

An automated telephone information service spends either 3, 6, or 10 minutes with each caller. The proportion of calls for each service length is 30%, 45%, and 25%, respectively.We want to simulate these service times in a spreadsheet. Our actual purpose is to learn how to generate random samples from a discrete distribution, in preparation for the queueing, inventory, and other examples to follow. This also is an example of a Monte Carlo simulation. Table 2 shows the input specification and a portion of the simulation table (the first 4 callers),
taken from the spreadsheet model, "Example2.2ServiceTimes.xls". The input specification defines the probabilities for the service times, and also includes the cumulative probabilities. Cumulative probabilities always increase to 1.0; as we shall see, the generation method uses the cumulative probabilities.The method always starts with a random number and ends with the desired value,

Table 2 Input Specification and Simulation Table for Service Times

	A	B	C	D
4		**Service Time**	**Probability**	**Cumulative Probability**
5				
6				
7		3	0.30	0.30
8		6	0.45	0.75
9		10	0.25	1.00
10				
11		**Number of Callers=**		25
12		**Simulation Table**		
13		**Step**	**Activity**	
14				
15		**Caller**	**Service Time**	
16		1	6	
17		2	6	
18		3	10	
19		4	6	

In this case, a random service time with the specified distribution. The simulation table in the spreadsheet

shows the resulting frequency distribution (or histogram) of the generated service times for 25 callers.

Figure 28 illustrates how to transform a random number to a service time. Think of it as throwing a special dart at a special dart board; the dart will always land somewhere on the unit interval, with equal probability of landing at any of the points on the line between 0 and 1. The subinterval where the dart lands determines the value generated.

In Figure 28, the probabilities (at the top) are represented by non-overlapping segments on a unit interval, and must add up to 1. The cumulative probabilities (just below the line) are represented bynpoints on the line. The arrows represent the transformation. The algorithm works as follows: First, choose a random number, say R. If R is in the first interval, that is, if R is less than or equal to 0.30, the procedure generates a sample of 3 minutes for service time. Similarly, if R is between 0.30 and 0.75, the service time is 6 minutes, and finally, if R is greater than 0.75, then the service time is 10 minutes. This algorithm easily generalizes to any discrete distribution.To illustrate the procedure, we first generate 5 samples of random numbers, either in Excel (using RAND or Rnd01 or any other random number generator), or take 5 samples from a table of random numbers such as Table A.1 in the Appendix. Suppose the 5 samples are: 0.9871 0.0226 0.0008 0.2128 0.8586 After the transformation illustrated in Figure 1, the resulting service times are:

10 3 3 3 10

In a small sample (here, size 5), we cannot expect the observed frequencies of occurrence of each value to be close to the probabilities; however, with a large enough sample, the sample

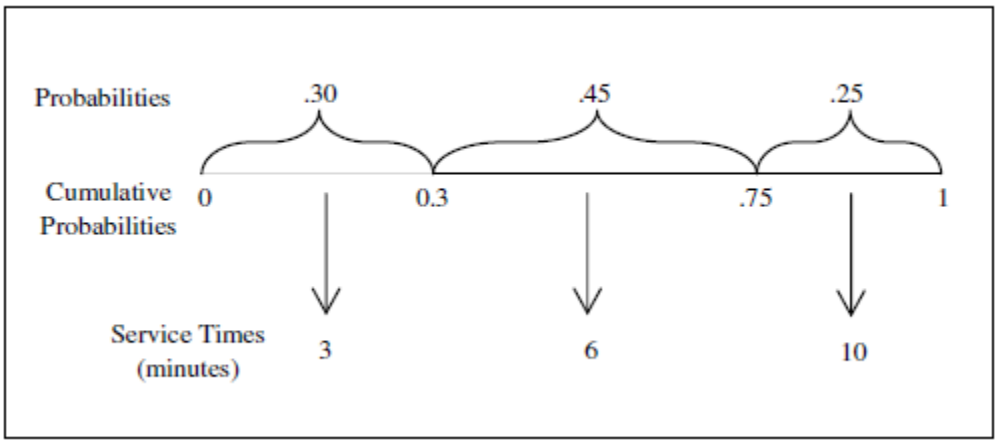


Figure 28 "Random Dart" Transformation from Random Number to Service Time

frequencies should be fairly close to the probabilities. You can experiment with the spreadsheet example, “Example2.2ServiceTimes.xls”, to see how close or far small samples can be from the assumed probabilities. In Excel, we might think we could implement this transformation, at least for distributions with a small number of values (here, 3), with a nested IF worksheet function, as in:

=IF(Rnd01()<=0.30,3,IF(Rnd01()<=0.75,6),10) (WRONG !!)

This attempt at a solution introduces a grave error. Each reference to the Rnd01() function generates a newrandom number; the transformation procedure depends on using just one random number. (You may want to modify the spreadsheet solution in “Example2.2ServiceTimes.xls” with this incorrect solution to see the resulting probabilities.) This approach can be modified to use two cells, the first to hold the result from Rnd01, the second with a modified IF statement.

A better approach is to use the VBA function called DiscreteEmp() (for discrete empirical, meaning defined by data) supplied with all the examples. To see how this function is used, see the spreadsheet solution, “Example2.2ServiceTimes.xls”; a typical cell for generating a random service time appears as follows:

=DiscreteEmp($D$7:$D$9, $B$7:$B$9)

where $D$7:$D$9 is the range of cells containing the cumulative probabilities and $B$7:$B$9 is the range of cells containing the desired values for service times. Discrete means that only the values listed will be generated. Each of the two worksheet ranges must be a column vector (meaning all data in one column), with each having the same length, that is, the same number of cells. The “$ in each address in front of each row and column indicates an absolute address rather than a relative address; this makes it easier to copy and paste the formula into all the cells in the “Service Time” column after typing it into just one cell.) probabilities and the second for the desired values. It works with any number of values in the distribution.