
Chapter 5

Arena Basics

The working simulation tool for the models in this book is *Arena*. Arena is a simulation environment consisting of module templates, built around *SIMAN* language constructs and other facilities, and augmented by a visual front end. This chapter provides an overview of Arena basics at an introductory level. For more detail, refer to Kelton et al. (2004). Because it may be hard to distinguish between Arena terms and generic terms, we shall always italicize any technical Arena terms throughout the book. Furthermore, we will adopt the notational convention that all characters in *SIMAN* constructs are always uppercase, while only the first character in Arena constructs is capitalized, and both are italicized.

SIMAN consists of two classes of objects: *blocks* and *elements*. More specifically, blocks are basic logic constructs that represent operations; for example, a *SEIZE* block models the seizing of a service facility by a transaction (referred to in Arena as “entity”), while a *RELEASE* block releases the facility for use by other transactions. *Elements* are objects that represent facilities, such as *RESOURCES* and *QUEUES*, or other components, such as *DSTATS* and *TALLIES*, used for statistics collection.

Arena's fundamental modeling components, called *modules*, are selected from template panels, such as *Basic Process*, *Advanced Process*, and *Advanced Transfer*,¹ and placed on a canvas in the course of model construction. A module is a high-level construct, composed of *SIMAN* blocks and/or elements. For example, a *Process* module models the processing of an entity, and internally consists of such blocks as *ASSIGN*, *QUEUE*, *SEIZE*, *DELAY*, and *RELEASE*. Arena also supports other modules, such as *Statistic*, *Variable*, and *Output* among many others. Frequently used Arena constructs (built-in variables and modules) are succinctly described in Appendix A.

Arena implements a programming paradigm that combines visual and textual programming. A typical Arena session involves the following activities:

1. Selecting module/block icons from a template panel, and placing them on a graphical model canvas (by drag and drop).

¹ Earlier versions of Arena have other panels, such as *Blocks*, *Elements*, *Common*, and *Support*. Later versions of Arena preserve these legacy panels in a template panel directory called *OldArena Templates*, and allow users to use them as necessary.

2. Connecting modules graphically to indicate physical flow paths of transactions and/or logical flow paths of control.
3. Parameterization of modules or elements using a text editor.
4. Writing code fragments in modules using a text editor. Arena code is case-insensitive; that is, upper and lower case letters are interchangeable.

Arena has a *graphical user interface* (GUI) built around the SIMAN language. In fact, simulation models can be built using SIMAN constructs from the *Blocks* and *Elements* template panels alone, since Arena modules are just subprograms written in SIMAN. Still, Arena is far more convenient than SIMAN, because it provides many handy features, such as high-level modules for model building, statistics definition and collection, animation of simulation runs (histories), and output report generation. Model building tends to be particularly intuitive, since many modules represent actual sub-systems in the conceptual model or the real-life system under study. Complex models usually require both Arena modules and SIMAN blocks.

All in all, Arena provides a module-oriented simulation environment to model practically any scenario involving the flow of transactions through a set of processes. Furthermore, while the modeler constructs a model interactively in both graphical and textual modes, Arena is busy in the background transcribing the whole model into SIMAN. Since Arena generates correct SIMAN code and checks the model for syntactic errors (graphical and textual), a large amount of initial debugging takes place automatically.

5.1 ARENA HOME SCREEN

An Arena home screen is shown in Figure 5.1.

The reader is encouraged to browse the Arena home screen, and examine the objects to be mentioned in the sequel.

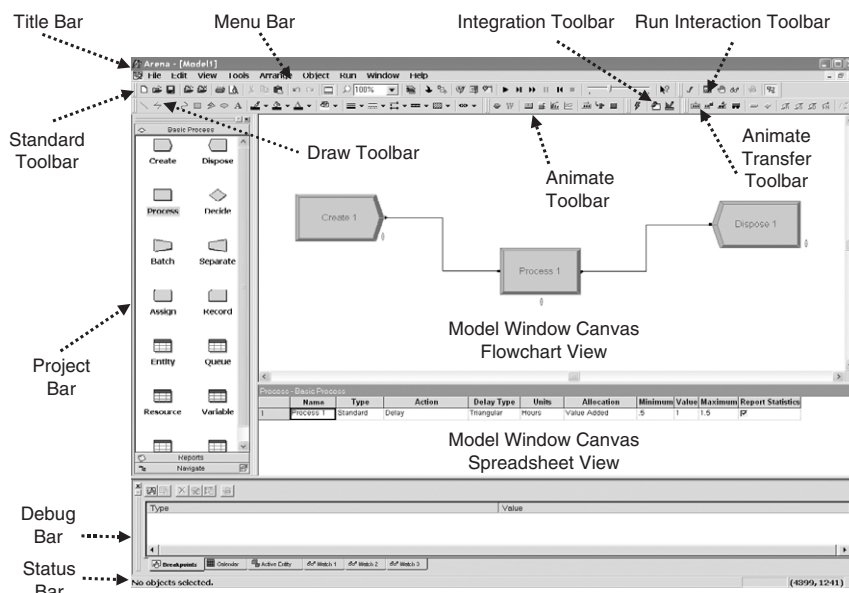


Figure 5.1 The Arena home screen.

The Arena home screen has a *Title* bar with the model name at its top. Below the *Title* bar is the *Arena Menu* bar, which consists of a set of general menus and Arena-specific menus. Below the *Menu* bar is a set of Arena toolbars that can be displayed and hidden by clicking the right mouse button on the background area of a specific toolbar. These toolbars consist of buttons that support model building and running; note that some toolbar buttons conveniently duplicate the functionality of certain menu options in the *Menu* bar.

The bulk of the home screen is allocated to a model window canvas consisting of a *flowchart* view and *spreadsheet* view. The user spawns modules and other objects, drags them into the flowchart view canvas, and progressively builds a model from component modules selected from the *Project* bar (on the left) with the aid of support functions. Selecting a module in the flowchart view pops up a spreadsheet view (below the flowchart view) that summarizes all module information and permits editing of the associated data.

In addition to the *Project* bar, the user can also pop up (or hide) two additional bars below the model window canvas. The *Debug* bar permits the user to track the evolution of simulation runs, while the *Status* bar (below it) displays instructions and feedback messages concerning user actions.

The *Menu* bar and the set of key Arena toolbars are described next. Consult the online help for complete descriptions of all Arena bars (menu bars, toolbars, and so on).

5.1.1 MENU BAR

The *Arena Menu* bar consists of a number of general menus—*File*, *Edit*, *View*, *Window*, and *Help*—which support quite generic functionality. It also has the following set of Arena-specific menus:

- The *Tools* menu provides access to simulation related tools and Arena parameters.
- The *Arrange* menu supports flowcharting and drawing operations.
- The *Object* menu supports module connections and submodel creation.
- The *Run* menu provides simulation run control. Its *Setup...* option opens a form that permits the user to enter information, such as project parameters (name, analyst, date, etc.), as well as replication parameters (length, warm-up period, etc.). It also has options that provide VCR-type functionality to run simulation replications and run control options to monitor entity motion, variable assignments, and so on.

5.1.2 PROJECT BAR

The *Project* bar lets the user access Arena template panels, where Arena modules, SIMAN blocks, and various other objects cohabit. Template panels can be attached to the *Project* bar by clicking the *Attach* button on the *Standard* toolbar. More specifically, when the *Attach* button is clicked, a dialog box pops up on the screen and shows the so-called *.tpo* files corresponding to each template panel. Choosing a *.tpo* file will attach its template panel to the *Project* bar. The Arena template panels available to users are as follows:

- The *Basic Process* template panel consists of a set of basic modules, such as *Create*, *Dispose*, *Process*, *Decide*, *Batch*, *Separate*, *Assign*, and *Record*.
- The *Advanced Process* template panel provides additional basic modules as well as more advanced ones, such as *Pickup*, *Dropoff*, and *Match*.
- The *Advanced Transfer* template panel consists of modules that support entity transfers in the model. These may be ordinary transfers or transfers using material-handling equipment.
- The *Reports* template panel supports report generation related to various components in a model, such as entities, resources, queues, and so on.
- The *Blocks* template panel contains the entire set of SIMAN blocks.
- The *Elements* template panel contains elements needed to declare model resources, queues, variables, attributes, and some statistics collection.

In addition to the Arena template panels above, the following Arena template panels from earlier versions are also supported:

- The *Common* template panel contains Arena modules such as *Arrive*, *Server*, *Depart*, *Inspect*, and so on, as well as element modules such as *Stats*, *Variables*, *Expressions*, and *Simulate*.
- The *Support* template panel contains a subset of frequently used SIMAN blocks.

The models we present in this book will mostly utilize modules from template panels of Arena 10.0 and later versions.

5.1.3 STANDARD TOOLBAR

The Arena *Standard* toolbar contains buttons that support model building. An important button in this bar is the *Connect* button, which supports visual programming. This button is used to connect Arena modules as well as SIMAN blocks, and the resulting diagram describes the flow of logical control. The *Time Patterns Editor* feature consists of three buttons that allow the modeler to schedule the availability of resources and their service rate. The *Standard* toolbar also provides VCR-style buttons to run an Arena model in interrupt mode to trace its evolution. More details on this VCR-like functionality are discussed in Section 6.3.1.

5.1.4 DRAW AND VIEW BARS

The *Draw* toolbar supports static drawing and coloring of Arena models. In a similar vein, the *View* toolbar (not shown in Figure 5.1) assists the user in viewing a model. Its buttons include *Zoom In*, *Zoom Out*, *View All*, and *View Previous*. These functions make it convenient to view large models at various levels of detail.

5.1.5 ANIMATE AND ANIMATE TRANSFER BARS

The *Animate* toolbar is used for animation (dynamic visualization) of Arena model objects during simulation runs. Animated objects include the simulation clock, queues, monitoring windows for variables, dynamic plots, and histogram functions. The

Animate Transfer toolbar is used to animate entity transfer activities, including materials handling (see Section 13.2 for more details).

5.1.6 RUN INTERACTION BAR

The *Run Interaction* toolbar supports run control functions to monitor simulation runs, such as access to SIMAN code and model debugging facilities. It also supports model visualization, such as the *Animate Connectors* button that switches on and off entity traffic animation over module connections. Because of this toolbar's fundamental role in model testing, it will be revisited in Chapter 6.

5.1.7 INTEGRATION BAR

The *Integration* toolbar supports data transfer (import and export) to other applications. It also permits Visual Basic programming and design. A primer on Visual Basic for Arena may be found in Appendix B.

5.1.8 DEBUG BAR

The *Debug* toolbar supports debugging of Arena models by monitoring and controlling the execution of a simulation run. It consists of two subwindows. The left subwindow can be used in command mode to set breakpoints, assign variable values, observe watched variables, and trace entity flows among modules. The right subwindow has tabs for viewing future SIMAN events in the model, displaying attributes of the current active entity, and watching user-defined Arena expressions. For more information, see Chapter 6.

5.2 EXAMPLE: A SIMPLE WORKSTATION

We now proceed to illustrate the basic features of Arena through a simple example. Consider a single workstation consisting of a machine with an infinite buffer in front of it. Jobs arrive randomly and wait in the buffer while the machine is busy. Eventually they are processed by the machine and leave the system. Job interarrival times are exponentially distributed with a mean of 30 minutes, while job processing times are exponentially distributed with a mean of 24 minutes. This system is known in queueing theory as the *M/M/1* queue (Kleinrock 1975).

The steady-state behavior of this system has been well studied, and analytical formulas have been derived for its main performance measures, such as distribution of the number of jobs in the system and average waiting time in the buffer (see *ibid.*) This example will compare the simulation statistics to their theoretical counterparts to gauge the accuracy of simulation results. Specifically, we shall estimate by an Arena simulation the average job delay in the buffer, the average number of jobs in the buffer, and machine utilization.

Simulating the above workstation calls for the following actions:

1. Jobs are created, one at a time, according to the prescribed interarrival distribution. Arriving jobs are dispatched to the workstation.

2. If the machine is busy processing another job, then the arriving job is queued in the buffer.
3. When a job advances to the head of the buffer, it seizes the machine for processing once it becomes available, and holds it for a time period sampled from the prescribed processing-time distribution.
4. On process completion, the job departs the machine and is removed from the system (but not before its contribution to the statistics of the requisite performance measures are computed).

A simple approach to modeling the workstation under study is depicted in Figure 5.2.

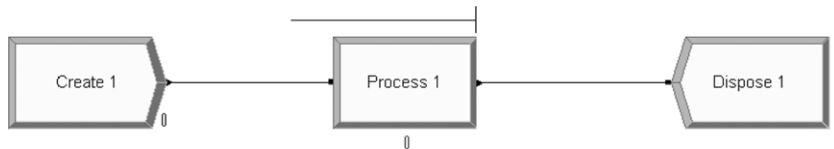


Figure 5.2 A simple Arena model of an $M/M/1$ queue.

In this model, jobs (Arena entities) are created by the *Create* module *Create 1*. Jobs then proceed to be processed in the *Process* module *Process 1*, after which they enter the *Dispose* module, *Dispose 1*, for removal from the model. The graphic shaped like an elongated T (called a *T-bar*) above the module *Process 1* represents space for waiting jobs (here, the workstation's buffer). The interarrival specification of the *Create* module is shown in the dialog box of Figure 5.3.

Figure 5.3 Dialog box for a *Create* module.

The dialog box contains information on job interarrival time (*Time Between Arrivals* section), batch size (*Entities per Arrival* field), maximal number of job arrivals (*Max Arrivals* field), time of first job creation (*First Creation* field), and so on. The *Type* pull-down menu in the *Time Between Arrivals* section offers the following options:

- *Random* (exponential interarrival times with mean given in the *Value* field)
- *Schedule* (allows the user to create arrival schedules using the *Schedule* module from the *Basic Process* template panel)

- *Constant* (specifies fixed interarrival times)
- *Expression* (any type of interarrival time pattern specified by an Arena expression, including Arena distributions)

The job processing mechanism (including priorities) is specified in the *Process* module, whose dialog box is shown in Figure 5.4.

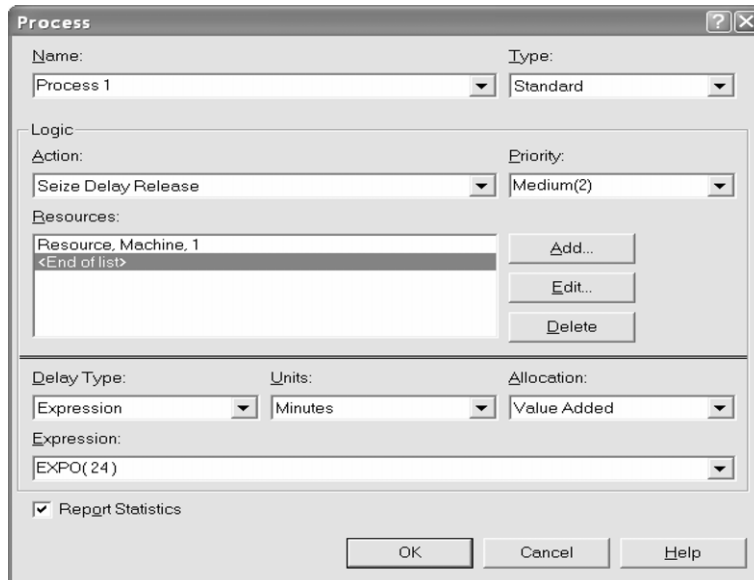
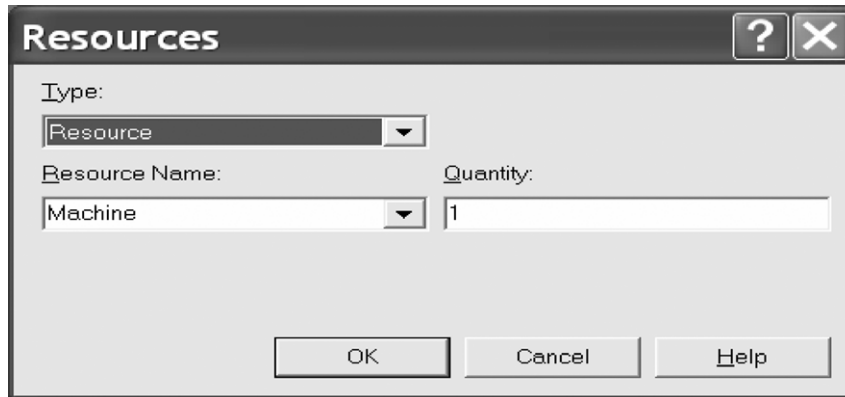


Figure 5.4 Dialog box for a *Process* module.

The *Action* field option, selected from the pull-down menu, is *Seize Delay Release*, which stands for a sequence of *SEIZE*, *DELAY*, and *RELEASE* SIMAN blocks. *SEIZE* and *RELEASE* blocks are used to model contention for a resource possessing a capacity (e.g., machines). When resource capacity is exhausted, the entities contending for the resource must wait until the resource is released. Thus the *SEIZE* block operates like a gate between entities and a resource. When the requisite quantity of resource becomes available, the gate opens and lets an entity seize the resource; otherwise, the gate bars the entity from seizing the resource until the requisite quantity becomes available. Note that the resource quantity seized should be an integer; otherwise Arena truncates it.

The processing (holding) time of a resource (*Machine* in our case) by an entity is specified via the *DELAY* block within the *Process* module. For instance, the dialog box of Figure 5.4 specifies that one unit of resource *Machine* be seized and held for an exponentially distributed time with a mean of 24 minutes. If multiple resources need to be seized simultaneously, all requisite resources are listed, and the holding time clock is started only after *all* requisite resources listed become available. When at least one of the listed resources is not available, arriving entities wait in a FIFO (First In First Out) queue, ordered further by priorities (specified in the *Priority* field of the dialog box). This discipline is called *FIFO within priority classes*, because higher-priority entities precede lower-priority ones, but arrivals within a given priority class are ordered FIFO.

The *Add* button in a *Process* module (see Figure 5.4) is used to specify resources and the resource quantities to be seized by an incoming entity. To this end, the *Add* button pops up a *Resources* dialog box as shown in Figure 5.5.



The **Resources** dialog box has a title bar with a question mark and a close button. It contains the following fields and buttons:

- Type:** A dropdown menu with "Resource" selected.
- Resource Name:** A dropdown menu with "Machine" selected.
- Quantity:** A text input field containing the value "1".
- Buttons:** "OK", "Cancel", and "Help" buttons at the bottom.

Figure 5.5 *Resources* dialog box for specifying a resource.

Note, however, that the capacity of resources introduced in a *Resources* dialog box is specified elsewhere, namely, in the *Resource* module spreadsheet (Figure 5.6).

Resource - Basic Process									
	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures	Report Statistics
1	Machine	Fixed Capacity	1	0.0	0.0	0.0		0 rows	<input checked="" type="checkbox"/>

Double-click here to add a new row.

Figure 5.6 *Resource* module spreadsheet for specifying resource capacities.

This spreadsheet is accessible in the *Basic Process* template of the *Project* bar, and is used to specify all resource capacities of an Arena model. The default capacity of resources is 1. More details on the *Resource* module spreadsheet are deferred to examples in the sequel.

The *Dispose* module implements an entity “sunset” mechanism, by simply discarding entities that enter it. As such, the *Dispose* module serves as a system “sink,” thereby counteracting the *Create* module, which serves as a system “source.” Note carefully that in the absence of properly placed *Dispose* modules, the number of entities created in the course of a simulation run can grow without bound, eventually exhausting available computer memory (or a prescribed limit) and terminating the run (very likely crashing the simulation program). A *Dispose* module is not necessary, however, when a fixed or bounded number of entities are created and circulate in the system indefinitely.

Whenever an Arena model is saved, the model is placed in a file with a *.doe* extension (e.g., *mymodel.doe*). Furthermore, whenever a model, such as *mymodel.doe*, is checked using the *Check Model* option in the *Run* menu or any run option in it, Arena automatically creates a number of files for internal use, including *mymodel.p* (program file), *mymodel.mdb* (Access database file), *mymodel.err* (errors file), *mymodel.opw* (model components file), and *mymodel.out* (SIMAN output report file). As these are internal Arena files, the modeler should not attempt to modify them.

Run control functionality is provided by the *Run* pull-down menu (see Figure 5.1). Selecting the *Setup...* option opens the *Run Setup* dialog box, which consists of multiple tabs, each with its own dialog box. In particular, the *Replication Parameters* tab permits the specification of the number of replications, replication length, and the warm-up period. (A warm-up period is a simulation of an initial interval, designed to “warm up” the system to a more representative state; this will be discussed in greater detail in Chapter 9 when we discuss output analysis.) In this example, the model will be simulated for 10,000 hours, with all other fields in the *Run Setup* dialog box retaining their default values.

The end result of a simulation run is a set of requisite statistics, such as mean waiting times, buffer size probabilities, and so on. These will be referred to as *run results*. Arena provides a considerable number of default statistics in a report, automatically generated at the end of a simulation run. Additional statistics can be obtained by adding statistics-collection modules to the model, such as *Record* (*Basic Process* template panel) and *Statistic* (*Advanced Process* template panel). However, when using SIMAN (the *Support* template panel in the *Old Arena Templates* folder), the user has to include additional blocks and elements in the model in order to affect statistics collection. The simpler statistics-collection facilities in Arena are one of the advantages of Arena over SIMAN. Subsequent chapters will provide additional information on statistics collection.

The run results of a single replication of the workstation model from Figure 5.2 are displayed in Figures 5.7 and 5.8.

3:33:44PM

Resources

September 1, 2005

A Simple Workstation

Replications: 1

Replication 1

Start Time: 0.00

Stop Time: 600,000.00

Time Units: Minutes

Machine

Usage	Value			
Total Number Seized	20,005.00			
Scheduled Utilization	0.8097			
Number Scheduled	1.0000	(Insufficient)	1.0000	1.0000
Number Busy	0.8097	0.014618405	0	1.0000
Instantaneous Utilization	0.8097	0.014618405	0	1.0000

Figure 5.7 Resource statistics from a single replication of the simple workstation model.

Figure 5.7 displays the *Resources* section, which includes resource utilization. The last three columns correspond to the half-width of the confidence interval (see Section 3.10), minimum observation, and maximum observation, respectively, of the corresponding statistics. The *(insufficient)* notation indicates that the number of observations is insufficient for adequate statistical confidence. Observe that the *Number Busy* item refers to the number of busy units of a resource, while the *Number Scheduled* item refers to resource capacity. The *Instantaneous Utilization* item pertains to utilization per

resource unit, namely, *Number Busy* divided by the *Number Scheduled*. We point out that in the long run, individual resource utilizations approach this number.

3:33:07PM

Queues

September 1, 2005

A Simple Workstation

Replications: 1

Replication 1

Start Time: 0.00

Stop Time: 600,000.00

Time Units: Minutes

Process 1.Queue

Time	Average	Half Width	Minimum	Maximum
Waiting Time	107.09	16.42523	0	807.06
Other	Average	Half Width	Minimum	Maximum
Number Waiting	3.5721	0.538999733	0	35.0000

Figure 5.8 Queue statistics from a single replication of the simple workstation model.

Figure 5.8 displays the *Queues* section, which consists of customer-oriented statistics (customer averages), such as mean waiting times in queues, as well as queue-oriented statistics, such as time averages of queue sizes (occupancies). Additional sections in an output report include *Frequencies* (probability estimates) and *User Specified* (any customized statistics). For more details on Arena statistics collection and output reporting, see Sections 5.4 and 5.5.

An examination of the run results (replication statistics) displayed in Figures 5.7 and 5.8 reveals that the machine utilization is about 81%. The average waiting time in the machine buffer is about 107 minutes, with a 95% confidence interval of 107.09 ± 16.42523 minutes, and a maximum of 807.06 minutes. The average number of jobs in the buffer is 3.57 jobs, with the maximal buffer length observed being 35 jobs.

Figure 5.7 can be used for preliminary verification of the workstation model. For example, we can use the classical formula $\rho = \lambda/\mu$ (Kleinrock 1975) of machine utilization to verify that the observed server utilization, ρ , is indeed the ratio of the job arrival rate ($\lambda = 1/30$) and the machine-processing rate ($\mu = 1/24$). Indeed, this ratio is 0.8, which is in close agreement with the run result, 0.8097. Note that this relation holds only approximately, due to the inherent variation in sampling simulation statistics. Other verification methods will be covered in some detail in Chapter 8.

5.3 ARENA DATA STORAGE OBJECTS

An important part of the model building process is assignment and storage of data supplied by the user (input parameters) or generated by the model during a simulation run (output observations). To this end, Arena provides three types of data storage objects: *variables*, *expressions*, and *attributes*. Variables and expressions can be introduced and initialized via the *Variable* and *Expression* spreadsheet modules, accessible from the *Basic Process* template and *Advanced Process* template, respectively, in the *Project* bar.

5.3.1 VARIABLES

Variables are user-defined *global* data storage objects used to store and modify state information at run initialization or in the course of a run. Such (global) variables are visible everywhere in the model; namely, they can be accessed, examined, and modified from every component of the model. In an Arena program, variables are typically examined in *Decide* modules and modified in *Assign* modules. Unlike user-defined variables, certain predefined Arena (system) variables are read-only (i.e., they may only be examined to decide on a course of action or to collect statistics), and cannot be assigned a new value by the user; the system is solely responsible for changing these values. For instance, the variable $NQ(Machine_Q)$ stores the current value of the number of entities in the queue called *Machine_Q*. Similarly, the variable $NR(Machine)$ stores the number of busy units of the resource called *Machine*. Other important Arena variables are *TNOW*, which stores the simulation run's current time (*simulation clock*), and *TFIN*, which stores the simulation completion time. A list of all Arena variables may be found in the *Arena Variables Guide* or in the *Arena Help* facility (see Section 6.6). Recall that a selected partial list appears in Appendix A.

5.3.2 EXPRESSIONS

Expressions can be viewed as specialized *variables* that store the value of an associated formula (expression). They are used as convenient shorthand to compute mathematical expressions that may recur in multiple parts of the model. Whenever an expression name is encountered in the model, it is promptly evaluated at that point in simulation time, and the computed value is substituted for the expression name. Variables of any kind (user defined or system defined) as well as attributes may be used in expressions.

5.3.3 ATTRIBUTES

Attributes are data storage objects associated with entities. Unlike variables, which are global, attributes are local to entities in the sense that each instance of an entity has its own copy of attributes. For example, a customer's arrival time can be stored in a customer attribute to allow the computation of individual waiting times. When arrivals consist of multiple types of customer, the type of an arrival can also be stored in a customer entity's attribute to allow separate statistics collection for each customer type.

5.4 ARENA OUTPUT STATISTICS COLLECTION

As mentioned before, the end product of a simulation is a set of statistics that estimate performance measures of the system under study. Recall that such statistics can be classified into the two standard categories of time averages and customer averages (see, e.g., Section 2.3). More specifically, *time averages* are obtained by dividing the area under the performance function (e.g., number in the system, periods of busy and idle states, etc.) by the elapsed simulation time. *Customer average* statistics are averages of customer-related performance values (e.g., customer waiting times in queues).

Arena provides two basic mechanisms for collecting simulation output statistics: one via the *Statistic* module, and the other via the *Record* module. Time average statistics are collected in Arena via the *Statistic* module, while customer-average statistics must be collected via a *Record* module, and (optionally) specified in the *Statistic* module. Arena statistics collection mechanisms are described next in some detail.

5.4.1 STATISTICS COLLECTION VIA THE *STATISTIC* MODULE

Detailed statistics collection in Arena is typically specified in the *Statistic* module located in the *Advanced Process* template panel. Selecting the *Statistic* module opens a dialog box. The modeler can then define statistics as rows of information in the spreadsheet view that lists all user-defined statistics. For each statistic, the modeler specifies a name in the *Name* column, and selects the type of statistic from a drop-down list in the *Type* column. The options are as follows:

Time-Persistent statistics are simply time average statistics in Arena terminology.

Typical *Time-Persistent* statistics are average queue lengths, server utilization, and various probabilities. Any user-defined probability or time average of an expression can be estimated using this option.

Tally statistics are customer averages, and have to be specified in a *Record* module (see Section 5.4.2) in order to initiate statistics collection. However, it is advisable to include the definition in the *Statistic* module as well, so that the entire set of statistics can be viewed in the same spreadsheet for modeling convenience.

Counter statistics are used to keep track of counts, and like the *Tally* option, have to be specified in a *Record* module (see Section 5.4.2) in order to initiate statistics collection.

Output statistics are obtained by evaluating an expression at the end of a simulation run. Expressions may involve Arena variables such as *DAVG(S)* (time average of the *Time-Persistent* statistic *S*), *TAVG(S)* (the average of *Tally* statistic *S*), *TFIN* (simulation completion time), *NR()*, *NQ()*, or any variable from the *Arena Variables Guide*.

Frequency statistics are used to produce frequency distributions of (random) expressions, such as Arena variables or resource states. This mechanism allows users to estimate steady-state probabilities of events, such as queue occupancy or resource states.

Note that all statistics defined in the *Statistic* module are reported automatically in the *User Specified* section of the Arena output report (see Section 5.5). Furthermore, *Queue* and *Resource Time-Persistent* statistics will be automatically computed and need not be defined in the *Statistic* module.

5.4.2 STATISTICS COLLECTION VIA THE *RECORD* MODULE

As mentioned in Section 5.2, the *Record* module is used to collect various statistics. Any statistics related to customer averages or customer observations, such as *Tally* and *Counter*, have to be specified in a *Record* module. Figure 5.9 displays a dialog box for a *Record* module, listing all types of statistics as options in the *Type* field.

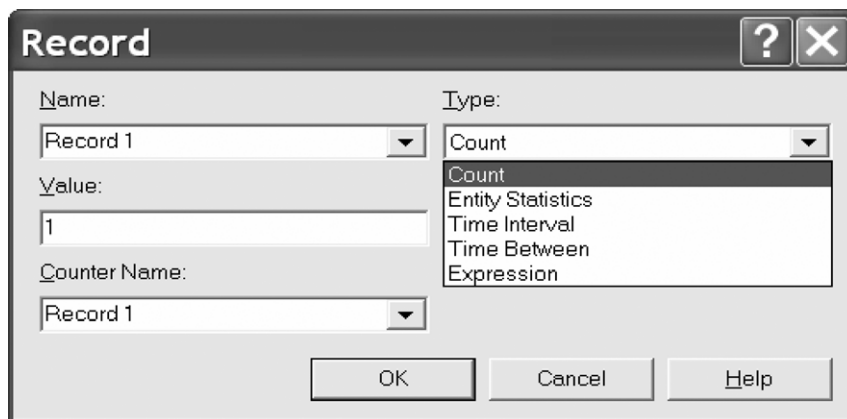


Figure 5.9 Types of statistics collected by the *Record* module.

These options are as follows:

1. The *Count* option maintains a count with a prescribed increment (positive or negative). The increment is quite general: it may be defined as any expression or function and may assume any real value. The corresponding counter is incremented whenever an entity enters the *Record* module.
2. The *Entity Statistics* option provides information on entities, such as time and costing/duration information.
3. The *Time Interval* option tallies the difference between the current time and the time stored in a prescribed attribute of the entering entity.
4. The *Time Between* option tallies the time interval between consecutive entries of entities in the *Record* module. These intervals correspond to interdeparture times from the module, and the reciprocal of the mean interdeparture times is the module's *throughput*.
5. Finally, the *Expression* option tallies an expression whose value is recomputed whenever an entity enters the *Record* module.

Note that except for the *Entity Statistics* option, all the options above are implemented in SIMAN via *COUNT* or *TALLY* blocks.

5.5 ARENA SIMULATION AND OUTPUT REPORTS

An Arena model run can be initiated and managed in two ways:

- Via the VCR-like buttons on the Arena *Standard* toolbar
- Via corresponding options in the *Run* pull-down menu bar

Refer to Section 6.3 for more details.

Standard Arena output reports provide summaries of simulation run statistics, as requested by the modeler implicitly or explicitly. Accordingly, Arena output reports fall into two categories:

Automatic reports. A number of Arena constructs, such as entities, queues, and resources, will automatically generate reports of summary statistics at the end of

a simulation run. Those statistics are implicitly specified by the modeler simply by dragging and dropping those modules into an Arena model, and no further action is required of the user.

User-specified reports. Reports on additional statistics can be obtained by explicitly specifying statistics collection via the *Statistic* module (*Advanced Process* template panel) and the *Record* module (*Basic Process* template panel). Recall that the *Statistic* module is specified in a spreadsheet view, while the *Record* module must be placed in the appropriate location in the model.

To request a formatted report, the user opens the *Reports* panel in the *Project* bar to display a list of report options that correspond to various types of statistics as follows:

- *Entities* reports automatically provide various entity counts.
- *Frequencies* reports provide time averages of expressions (including probabilities as a special case). The expression must be specified in a *Statistic* module with the *Frequency* option selected.
- *Processes* reports provide statistics associated with each *Process* module. These include incoming and outgoing entity counts, average service times, and average delays. Time-oriented statistics (e.g., delays) include the average, half-width of 95% confidence intervals, and minimal and maximal observed values. The half-width value is not displayed if the observations are correlated or if there are insufficient data (in which case Arena prints (*insufficient*) in the corresponding field).
- *Queues* reports provide statistics for each queue in the model, such as average queue delay and average queue size. Additional statistics include the half-width of the 95% confidence interval, and minimal and maximal observed values.
- *Resources* reports provide statistics for each resource in the model, such as utilization, average number of busy resource units, and number of times seized.
- *User Specified* reports are generated in response to explicit modeler requests for statistics collection in the *Statistic* module or *Record* modules (see Section 5.4). These include any *Time-Persistent*, *Tally*, *Count*, *Frequency*, and *Output* statistics.

Clicking on a report option in the *Reports* template panel of the *Project* bar prompts Arena to format the corresponding report and display it in a window. The top of that window contains VCR-like navigation buttons allowing the user to navigate report pages. Any number of options may be selected (sequentially), and the resultant report generated by Arena will consist of corresponding report sections. The *Toggle Group Tree* button at the top of the report window pops up a *Preview* panel to aid in navigating report sections.

5.6 EXAMPLE: TWO PROCESSES IN SERIES

This section presents a two-stage manufacturing model with two processes in series. Jobs arrive at an assembly workstation with exponentially distributed interarrival times of mean 5 hours. We assume that the assembly process has all the raw materials necessary to carry out the operation. The assembly time is uniformly distributed between 2 and 6 hours. After the process is completed, a quality control inspection is performed, and past data reveal that 15% of the jobs fail any given inspection and go back to the assembly operation for rework (jobs may end up going through multiple reworks until they pass inspection). Jobs that pass inspection go to the next stage, which

is a painting operation that takes 3 hours per job. We are interested in simulating the system for 100,000 hours to obtain process utilizations, average number of reworks per job, average job waiting times, and average job flow times (elapsed times experienced by job entities). Figure 5.10 depicts the corresponding Arena model and a snapshot of its state.

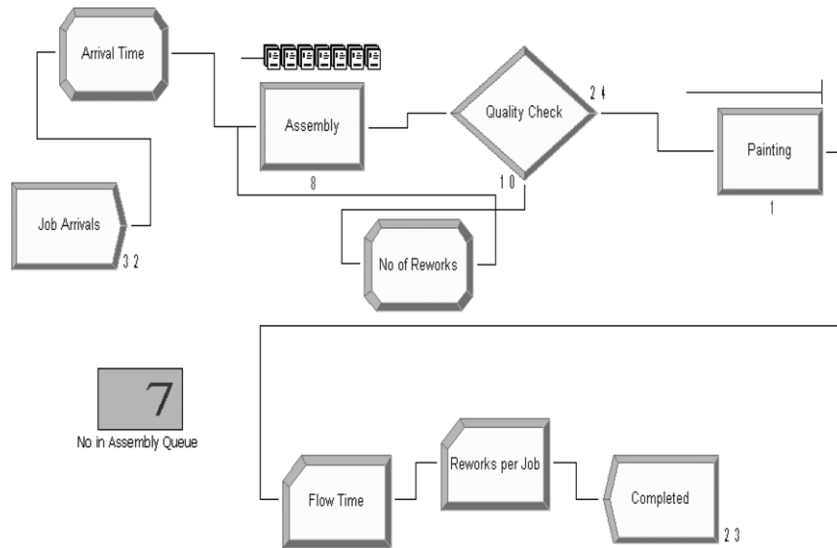


Figure 5.10 Arena model of two processes in series.

In this Arena model, entities represent jobs that are created by the *Create* module *Job Arrivals*, and enter the *Assign* module *Arrival Time*, where their arrival time is assigned to an attribute called *ArrTime*. Job entities then proceed to the *Process* module *Assembly*, where they undergo assembly. Next, assembled job entities go through inspection in the *Decide* module *Quality Check*, and those needing rework are routed to the *Assign* module *Number of Reworks* to record the number of reworks per job, and then back to module *Assembly*. Job entities that pass inspection proceed to be painted in the *Process* module *Painting*, which completes the manufacturing operation. Completed job entities go through *Record* modules *Flow Time* and *Reworks per Job* to collect statistics of interest. Finally, job entities enter the *Dispose* module *Completed*, where they are removed from the model.

The numbers at bottom right of the module icons display the number of entities that departed from the module. The icons above the module *Assembly* represent jobs waiting in the *Assembly* queue, called *Assembly.Queue*. The *Variable* window at bottom left of the figure displays the number of the aforementioned jobs. Job icons and data displayed on the module are *dynamic*, that is, they change as the simulation run unfolds in time. For example, this snapshot indicates that at the time it was taken, 28 jobs were created, 20 jobs were completed, 7 jobs were waiting to be assembled, and 1 job was being assembled.

We next proceed to explain the modules used in the model in more detail (note that the default module names have been replaced here by more expressive names relevant to the problem at hand).

The arrival time assignment is carried out in the *Assign* module *Arrival Time*, whose dialog boxes are shown in Figure 5.11.

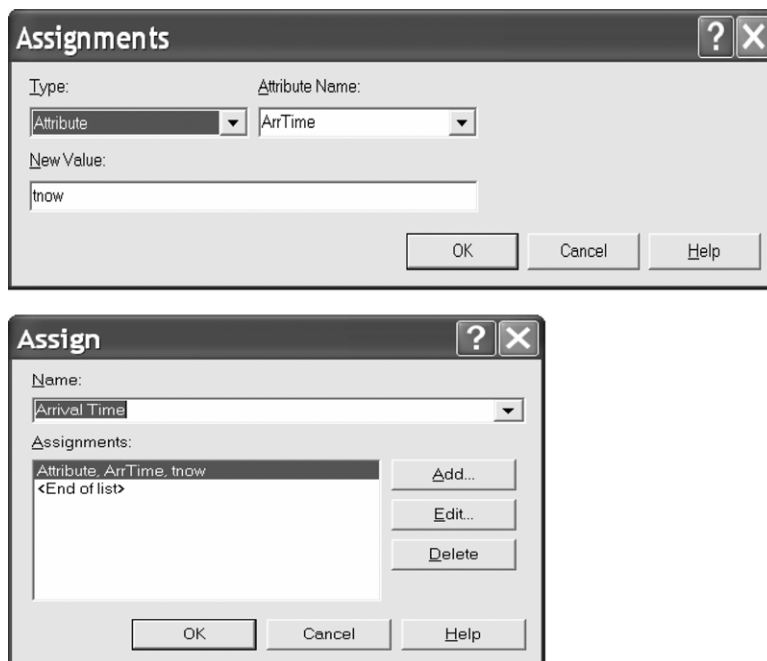


Figure 5.11 Dialog boxes for assigning an arrival time to a job entity's attribute.

Note that the bottom dialog box appears first on the screen, while clicking on an item in its *Assignments* field pops up the top dialog box for entering a value for the corresponding variable or attribute.

The module *Assembly* in Figure 5.10 is, in fact, a *Process* module that models the assembly operation with the appropriate job assembly time specification. Recall that the T-bar above the *Process* icon represents an animated *Queue* object for holding entities waiting for an opportune condition (e.g., to seize a resource). Note that the T-bar is displayed in a *Process* module only if the module contains a *Seize* option. The dialog box of the module *Assembly* in Figure 5.10 is displayed in Figure 5.12.

A *Decide* module, called *Quality Check*, follows the assembly operation, representing a quality control check. The *Decide* dialog box is shown in Figure 5.13. This module executes entity transfers, which may be probabilistic or based on the truth or falsity of some logical condition. Our example assumes that transfer times between processes are negligible, and therefore instantaneous. Here, we have a *two-way probabilistic branching* that splits the incoming stream of job entities into two outgoing streams:

With probability 0.85, an incoming job is deemed “good” (passed inspection), and is forwarded to the *Process* module, called *Painting*.

With probability 0.15 an incoming job is deemed “bad” (failed inspection), and is routed back to the *Process* module, called *Assembly*, for rework.

For convenience, the *Type* field provides separate options for simple two-way branching, which is the most common, as well as general *n-way* branching, which is more complex. In *n-way branching*, this module has multiple exits, exactly one

The screenshot shows the 'Process' dialog box for the 'Assembly' module. The 'Name' field is set to 'Assembly' and the 'Type' is 'Standard'. Under the 'Logic' section, the 'Action' is 'Seize Delay Release' and the 'Priority' is 'High(1)'. The 'Resources' list contains 'Resource, Assembler, 1' and '<End of list>'. There are 'Add...', 'Edit...', and 'Delete' buttons for the resources. The 'Delay Type' is 'Uniform', 'Units' are 'Hours', and 'Allocation' is 'Value Added'. The 'Minimum' value is 2 and the 'Maximum' value is 6. The 'Report Statistics' checkbox is checked. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

Figure 5.12 Dialog box of the *Process* module *Assembly*.

The screenshot shows the 'Decide' dialog box for the 'Quality Check' module. The 'Name' field is 'Quality Check' and the 'Type' is '2-way by Chance'. The 'Percent True (0-100)' is set to 85%. A dropdown menu for 'Type' is open, showing options: '2-way by Chance', '2-way by Condition', 'N-way by Chance', and 'N-way by Condition'. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

Figure 5.13 Dialog box of the *Decide* module *Quality Check* with a two-way probabilistic branching.

of which is designated the *else* branch and serves for default exits of entities from the module. More specifically, in *probabilistic branching*, a branch is taken with its associated probability, and the *else* branch automatically complements the exit probabilities, ensuring that they sum to 1. In *conditional branching*, the entity exits at the first branch whose condition evaluates to true, and if all conditions are false, the entity exits at the *else* branch.

Job entities that fail inspection enter the *Assign* module, called *No of Reworks*, where the job attribute, called *Total Reworks*, is incremented by the expression $Total\ Reworks + 1$. The dialog box for this module is shown in Figure 5.14.

Job entities departing from the *Painting* module enter two *Record* modules called *Flow Time* and *Reworks per Job*, to record their flow times and total number of reworks per job, respectively. In our example, the flow time is the difference between the job

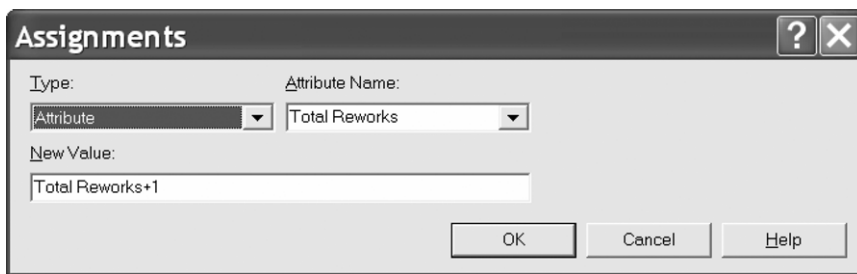


Figure 5.14 Dialog box of the *Assign* module *No of Reworks*.

departure time from the *Painting* module and the job's arrival time at module *Assembly*. Note that the job flow time includes any delays in queues as well as processing times. The average flow time is a customer average (*Time Interval* statistic), computed internally by a *TALLY* block, which can be verified by accessing the corresponding *.mod* file. The dialog box of the *Record* module is shown in Figure 5.15.

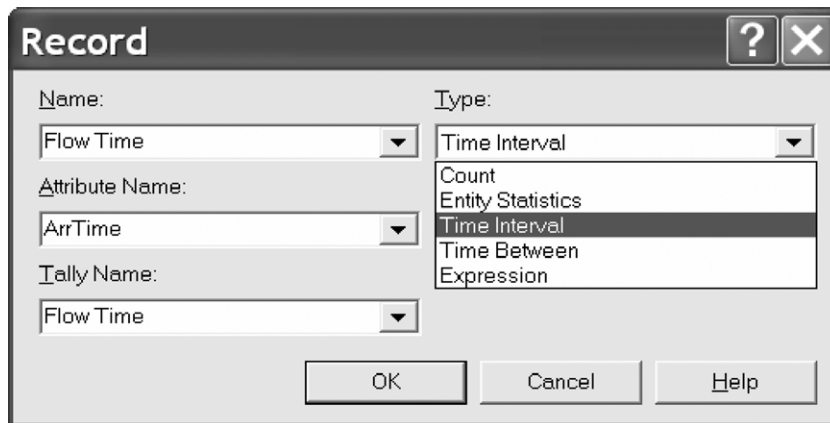


Figure 5.15 Dialog box of a *Record* module tallying job flow times.

In the next *Record* module, the expression consisting of the job entity's attribute *Total Reworks* is tallied in the *Record* module called *Reworks per Job*, as shown in the dialog box of Figure 5.16.

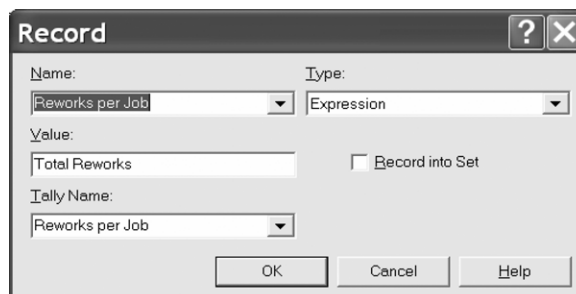


Figure 5.16 Dialog box of the *Record* module tallying the number of reworks per job.

Finally, job entities proceed to be disposed of in the *Dispose* module, called *Completed*. A single replication of the model was run for 100,000 hours, and the results are displayed in Figures 5.17, 5.18, and 5.19.

2:53:05PM

Resources

September 8, 2005

Assembly Op.

Replications: 1

Replication 1

Start Time: 0.00

Stop Time: 100,000.00

Time Units: Hours

Resource Detail Summary

Usage

	<u>Inst Util</u>	<u>Num Busy</u>	<u>Num Sched</u>	<u>Num Seized</u>	<u>Sched Util</u>
Assembler	0.94	0.94	1.00	23,417.00	0.94
Painter	0.59	0.59	1.00	19,830.00	0.59

Figure 5.17 Resource statistics for the manufacturing model (two processes in series).

2:53:41PM

Queues

September 8, 2005

Assembly Op.

Replications: 1

Replication 1

Start Time: 0.00

Stop Time: 100,000.00

Time Units: Hours

Queue Detail Summary

Time

	<u>Waiting Time</u>
Assembly.Queue	35.28
Painting.Queue	0.14

Other

	<u>Number Waiting</u>
Assembly.Queue	8.27
Painting.Queue	0.03

Figure 5.18 Queue statistics for the manufacturing model (two processes in series).

2:54:57PM

User Specified

September 8, 2005

Assembly Op.

Replications: 1

Replication 1

Start Time: 0.00

Stop Time: 100,000.00

Time Units: Hours

Tally

Expression	Average	Half Width	Minimum	Maximum
Reworks per Job	0.1807	0.006949558	0	4.0000
Interval	Average	Half Width	Minimum	Maximum
Flow Time	49.5252	(Correlated)	5.0004	926.57

Figure 5.19 Flow time statistics for the manufacturing model (two processes in series).

Figure 5.17 shows that the utilization estimates of the *Assembler* resource at the *Assembly* module and the *Painter* resource at the *Painting* module are 0.94 and 0.59, respectively. These estimates in Figure 5.17 correspond to a heavy traffic regime in the former and a medium traffic regime in the latter. These result in long average buffer delays and large average buffer occupancy in the assembly process, and in short average buffer delays and low average buffer occupancy in the painting process as shown in Figure 5.18. Finally, the *Tally* section of the *User Specified* output in Figure 5.19 displays not only averages, but also the corresponding 95% confidence interval half-widths, as well as the minimal and maximal observations for the number of reworks and flow time per job. Note that some jobs underwent as many as four reworks, while the average number of reworks is just 0.18, indicating a low level of reworks. The average flow time (49.5252 hours) is moderately longer than the sum of the average delays in the *Assembly* and *Painting* queues (35.42 hours) and the sum of average processing times there (7 hours), due to additional rework performed at module *Assembly*.

5.7 EXAMPLE: A HOSPITAL EMERGENCY ROOM

This section presents a more detailed model—in this case, of an emergency room in a small hospital—to further illustrate the power of simulation modeling.

5.7.1 PROBLEM STATEMENT

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 5.20 depicts a diagram of patient sojourn in the emergency room system, from arrival to discharge.

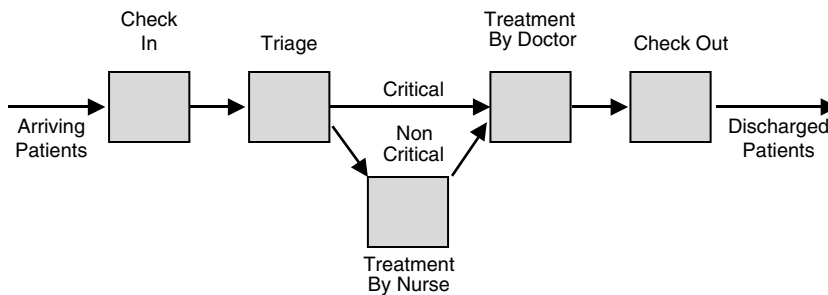


Figure 5.20 Patient sojourn in a hospital emergency room system.

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 first undergoes *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

To estimate the requisite statistics, the hospital emergency room was simulated for a period of 1 year.

5.7.2 ARENA MODEL

Having studied the problem statement, we now proceed to construct an Arena model of the system under study. Figure 5.21 depicts an Arena model of the 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

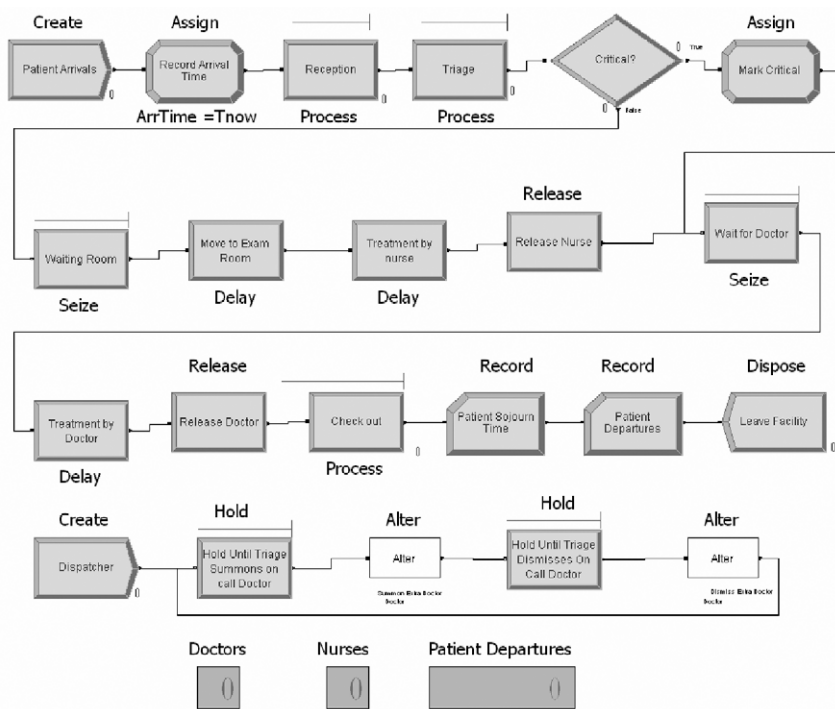


Figure 5.21 Arena model for the emergency room system.

summary reports. We assume that the emergency room is initially empty and the on-call doctor is not present.

We next proceed to examine the Arena model logic of Figure 5.21 in some detail, including the role of common modules from the *Basic Process* and *Advanced Process* template panels.

5.7.3 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 5.22 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).

Create

Name: Patient Arrivals Entity Type: Patient

Time Between Arrivals
 Type: Random (Expo) Value: 10 Units: Minutes

Entities per Arrival: 1 Max Arrivals: Infinite First Creation: 0.0

OK Cancel Help

Figure 5.22 Dialog box of the *Create* module *Patient Arrivals*.

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 5.23.

Process

Name: Reception Type: Standard

Logic
 Action: Seize Delay Release Priority: Medium(2)

Resources:
 Resource, Receptionist, 1
 <End of list>

Add... Edit... Delete...

Delay Type: Uniform Units: Minutes Allocation: Value Added

Minimum: 6 Maximum: 12

☒ Report Statistics

OK Cancel Help

Figure 5.23 Dialog box of the *Process* module *Reception*.

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 5.24 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.

The screenshot shows the 'Process' dialog box for the 'Triage' module. The 'Name' field is set to 'Triage' and the 'Type' is 'Standard'. Under the 'Logic' section, the 'Action' is 'Seize Delay Release' and the 'Priority' is 'Medium(2)'. The 'Resources' list contains 'Resource, Doctor, 1'. Below the resources, there are buttons for 'Add...', 'Edit...', and 'Delete'. The 'Delay Type' is 'Triangular', 'Units' are 'Minutes', and 'Allocation' is 'Value Added'. The 'Minimum' is 3, 'Value (Most Likely)' is 5, and 'Maximum' is 15. The 'Report Statistics' checkbox is checked. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

Figure 5.24 Dialog box of the *Process* module *Triage*.

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 5.25.

In Figure 5.25, the *2-Way by Chance* option specifies a two-way random branching based on the result of a random experiment, and the *Percent True* field indicates that 40% of the time the result is true (so 60% of the time the result is false). Accordingly, the corresponding branches emanating from the *Decide* module *Critical?* in Figure 5.21 are labeled *true* and *false*. Thus, a patient entity emerging from module *Critical?* takes either the *true* branch or the *false* branch as follows:

- 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.

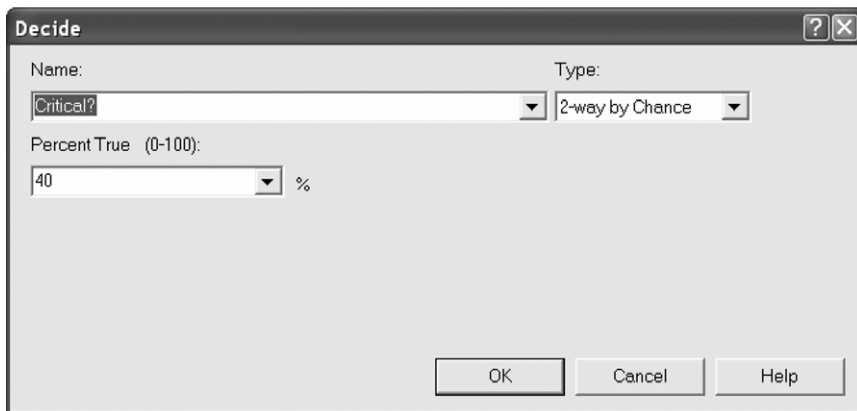


Figure 5.25 Dialog box of the *Decide* module *Critical?* for determining patient criticality.

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 5.26.

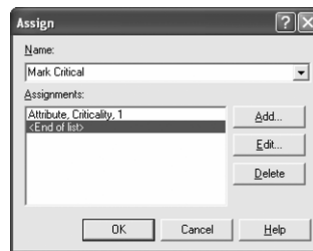


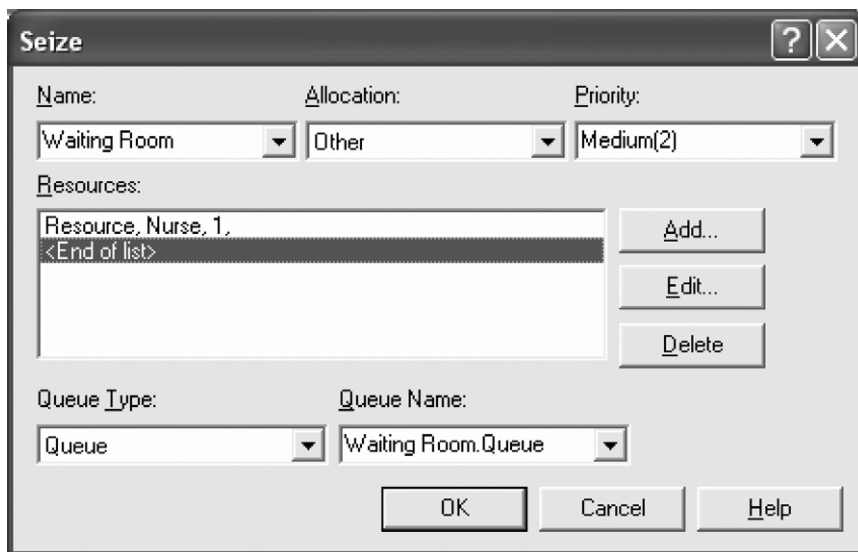
Figure 5.26 Dialog box of the *Assign* module *Mark Critical* for marking critical patients.

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 5.27.

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 5.28 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 5.29.

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*



Seize [?] [X]

Name: Allocation: Priority:

Resources:

Resource, Nurse, 1,	Add...
<End of list>	

Edit...
Delete

Queue Type: Queue Name:

OK Cancel Help

Figure 5.27 Dialog box of the *Seize* module *Waiting Room*.



Delay [?] [X]

Name: Allocation:

Delay Time: Units:

OK Cancel Help

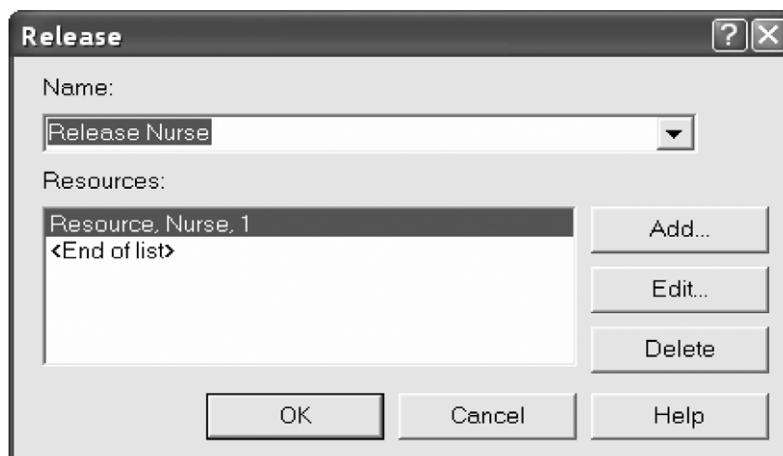
Delay [?] [X]

Name: Allocation:

Delay Time: Units:

OK Cancel Help

Figure 5.28 Dialog boxes of the *Delay* modules *Move to Treatment Room* (left) and *Treatment by Nurse* (right).



Release [?] [X]

Name:

Resources:

Resource, Nurse, 1	Add...
<End of list>	

Edit...
Delete

OK Cancel Help

Figure 5.29 Dialog box of the *Release* module *Release Nurse*.

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). The dialog box of the *Wait for Doctor* module is shown in Figure 5.30.

The dialog box is titled "Seize". It contains the following fields and controls:

- Name:** A dropdown menu with "Wait for Doctor" selected.
- Allocation:** A dropdown menu with "Other" selected.
- Priority:** A dropdown menu with "Medium[2]" selected.
- Resources:** A list box containing "Resource, Doctor, 1," and "<End of list>". To the right are buttons for "Add...", "Edit...", and "Delete".
- Queue Type:** A dropdown menu with "Queue" selected.
- Queue Name:** A dropdown menu with "Wait for Doctor.Queue" selected.
- At the bottom are "OK", "Cancel", and "Help" buttons.

Figure 5.30 Dialog box of the *Seize* module *Wait for Doctor*.

All patient entities wait in the queue, called *Wait for Doctor.Queue* to seize an available doctor, with critical patient entities receiving priority in treatment over noncritical ones. To this end, all patient entities queue up *FIFO within priority classes*, that is, all critical patient entities precede all noncritical ones, but each patient category is queued in the order of arrival. This is achieved by specifying the appropriate queueing discipline in the *Queue* module, whose spreadsheet view is shown in Figure 5.31.

	Name	Type	Attribute Name	Shared	Report Statistics
1	Check out.Queue	First In First Out	Attribute 1	<input type="checkbox"/>	<input checked="" type="checkbox"/>
2	Triage.Queue	First In First Out	Attribute 1	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3	Reception.Queue	First In First Out	Attribute 1	<input type="checkbox"/>	<input checked="" type="checkbox"/>
4	Wait for Doctor.Queue	Highest Attribute Value	Criticality	<input type="checkbox"/>	<input checked="" type="checkbox"/>
5	Waiting Room.Queue	First In First Out	Attribute 1	<input type="checkbox"/>	<input checked="" type="checkbox"/>
6	Hold Until Triage Summons on call Doctor.Queue	First In First Out	Attribute 1	<input type="checkbox"/>	<input checked="" type="checkbox"/>
7	Hold Until Triage Dismisses On Call Doctor.Queue	First In First Out	Attribute 1	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Double-click here to add a new row.

Figure 5.31 Spreadsheet view of the *Queue* module specifying queueing disciplines.

Each row in the spreadsheet specifies a queue in the Arena model, while columns *Type* and *Attribute Name* specify jointly the queueing discipline. Observe that all rows, except row 4, specify the ordinary FIFO discipline, while row 4 implicitly specifies the *FIFO within priority classes* discipline. More specifically, patient entities in *Wait for Doctor.Queue* queue up FIFO, but their queueing priority is determined by their *Criticality* attribute (the higher the value of *Criticality*, the higher the priority).

Once a doctor becomes available, the patient entity at the head of the line seizes that doctor and proceeds to the *Delay* module, called *Treatment by Doctor*, whose dialog box is depicted in Figure 5.32.

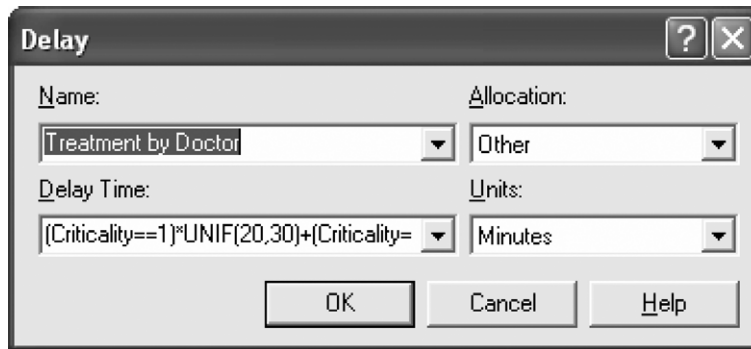


Figure 5.32 Dialog box of the *Delay* module *Treatment by Doctor*.

Recall that the treatment duration of a patient depends on its level of criticality, namely, on its *Criticality* attribute: for critical patients, the duration is uniform between 20 and 30 minutes, while for noncritical ones it is uniform between 5 and 10 minutes only. This dependence is captured in the *Delay Time* field of Figure 5.32 by the expression

$$(Criticality == 1) * UNIF(20, 30) + (Criticality == 0) * UNIF(5, 10).$$

Recall that $(Criticality == 1)$ and $(Criticality == 0)$ are logical expressions (predicates) that return 1 or 0 according to logical value of the expression in parentheses, which evaluates to true or false, respectively. Thus, for critical patients the expression above reduces to $UNIF(20, 30)$, whereas for noncritical ones it reduces to $UNIF(5, 10)$, which are the requisite distributions.

Following treatment by a doctor, all patient entities proceed to the *Release* module, called *Release Doctor*, where the doctor administering the treatment is released to other patients. Next, all patient entities are discharged from the emergency room. To this end, they enter the *Process* module, called *Check Out*, whose dialog box is shown in Figure 5.33. The checkout procedure requires a patient to seize a receptionist for a uniform time between 10 and 20 minutes, before releasing that receptionist.

Finally, patient entities enter two statistics-collecting *Record* modules, called *Patient Sojourn Time* and *Patient Departures*, respectively, whose dialog boxes are depicted in Figure 5.34.

The first *Record* module (left) tallies the total time that patient entities spend in the emergency room, from arrival to discharge (sojourn time)—a measure of patient satisfaction. This statistic is specified in the *Type* field by the *Time Interval* option, and is computed as $T_{now} - ArrTime$, namely, the difference between the current time and the patient's arrival time as recorded in its *ArrTime* attribute. The second *Record* module (right) simply counts the number of patient entities discharged from the emergency room—a measure of emergency room productivity. This statistic is specified in the *Type* field by the *Count* option, and is computed by incrementing a counter variable, called *Patient Departures* (see the *Counter Name* field), whenever a patient entity enters this module. Note that the modeler can access the current value of any counter via the Arena variable $NC(counter_name)$.

The **Process** dialog box for the **Check Out** module is shown. It includes the following fields and controls:

- Name:** Check Out (dropdown)
- Type:** Standard (dropdown)
- Logic:**
 - Action:** Seize Delay Release (dropdown)
 - Priority:** Medium(2) (dropdown)
 - Resources:**
 - Resource, Receptionist 1
 - <End of list>
 - Add...** button
 - Edit...** button
 - Delete** button
- Delay Type:** Uniform (dropdown)
- Units:** Minutes (dropdown)
- Allocation:** Value Added (dropdown)
- Minimum:** 10 (text box)
- Maximum:** 20 (text box)
- ☒ **Report Statistics**
- OK**, **Cancel**, and **Help** buttons at the bottom.

Figure 5.33 Dialog box of the *Process* module *Check Out*.

Two **Record** dialog boxes are shown side-by-side:

- Left Dialog (Patient Sojourn Time):**
 - Name:** Patient Sojourn Time (dropdown)
 - Type:** Time Interval (dropdown)
 - Attribute Name:** ArrTime (dropdown)
 - ☐ **Record into Set**
 - Tally Name:** Patient Sojourn Time (dropdown)
 - OK**, **Cancel**, and **Help** buttons.
- Right Dialog (Patient Departures):**
 - Name:** Patient Departures (dropdown)
 - Type:** Count (dropdown)
 - Value:** 1 (text box)
 - ☐ **Record into Set**
 - Counter Name:** Patient Departures (dropdown)
 - OK**, **Cancel**, and **Help** buttons.

Figure 5.34 Dialog boxes of the *Record* modules *Patient Sojourn Time* (left) and *Patient Departures* (right).

At long last, patient entities enter the *Dispose* module, called *Leave Facility*, after which they are removed from the model.

5.7.4 ON-CALL DOCTOR SEGMENT

The logic of this segment is controlled by a single circulating entity, dubbed *administrator*. The idea is to have the administrator modulate the number of doctors in the emergency room, depending on the number of patients in triage. Recall that emergency room doctors are a resource, so the administrator need only change this resource capacity as prevailing conditions change in the triage operation.

First, a single administrator entity is created at time 0, as shown in the dialog box of the *Create* module, called *Dispatcher*, in Figure 5.35.

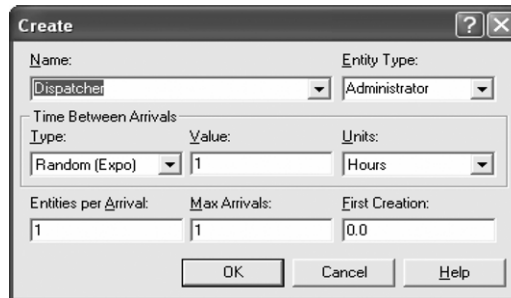


Figure 5.35 Dialog box of the *Create* module *Dispatcher*.

Note that this module simply creates precisely one entity of type *Administrator* at time 0. Consequently, the arrival stream specified by this module stops after the first entity is created, and therefore the interarrival time specification in the *Type* and *Value* fields is immaterial.

Since at this point the emergency room does not have the on-call doctor on duty, the administrator next watches for the condition that triggers summoning of the on-call doctor. To this end, the administrator enters the *Hold* module, called *Hold Until Triage Summons On-Call Doctor*, whose dialog box is shown in Figure 5.36.

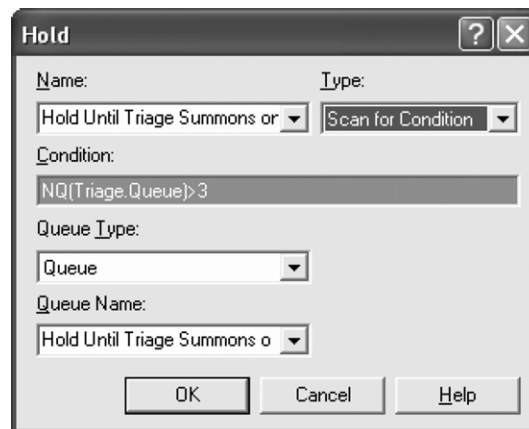


Figure 5.36 Dialog box of the *Hold* module *Hold Until Triage Summons On-Call Doctor*.

The administrator entity is held in this module (actually in a queue contained in this module as indicated by the *Queue Name* field) until the number of patients in triage exceeds three, signaling that the time has come to summon the on-call doctor. To watch for this condition, the *Scan for Condition* option is selected in the *Type* field, and the condition triggering the summoning of the on-call doctor is specified in the *Condition* field as

$$NQ(Triage.Queue) > 3,$$

where $NQ(Triage.Queue)$ is the Arena variable that holds the current number of entities in the queue. As soon as this condition becomes true, the administrator entity proceeds

to perform the action of summoning the on-call doctor by entering the *Alter* module, labeled *Summon Extra Doctor*, whose dialog box is shown in Figure 5.37.

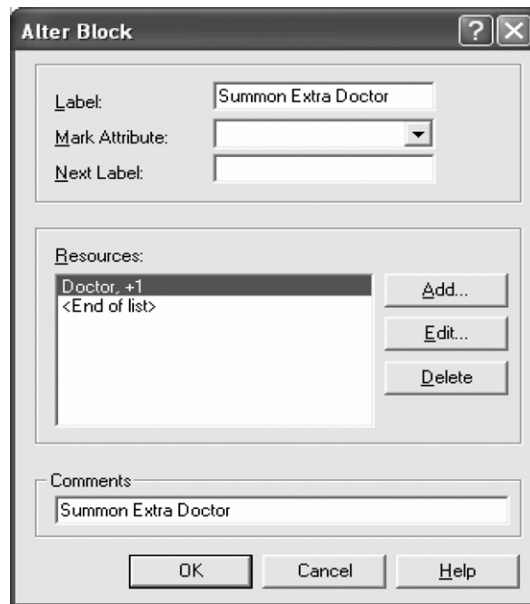


Figure 5.37 Dialog box of the *Alter* block *Summon Extra Doctor*.

Note that this module is called a *block* here, which is Arena's old term for *module*. To provide backward compatibility with older versions, Arena maintains a set of old blocks, which may be selected from the *Blocks* template panel, *Alter* included. The *Alter* block is used to alter model parameters. In our model, the administrator entity entering this block causes the *Doctor* resource pool to be incremented by 1, as evidenced by the expression in the *Resources* field above. This has the effect of increasing the available number of doctors in the emergency room by 1 (note that doctors do not have individual identities in our model).

The administrator entity next watches for the condition that triggers dismissal of the on-call doctor. To this end, it proceeds to the *Hold* module, called *Hold Until Triage Dismisses On-Call Doctor*, whose dialog box is depicted in Figure 5.38.

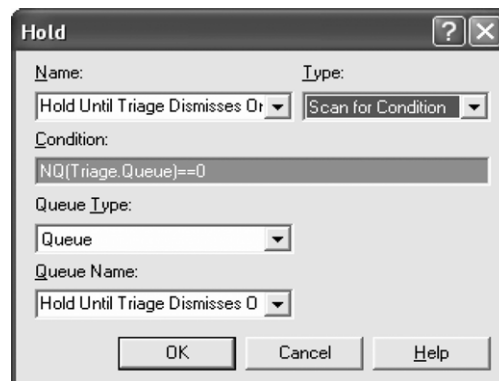


Figure 5.38 Dialog box of the *Hold* module *Hold Until Triage Dismisses On-Call Doctor*.

In a vein similar to the summoning action, the administrator entity is held in this module until the number of patient entities in triage drops to 0, signaling that the time has come to dismiss the on-call doctor, as evidenced by the *Condition* field expression.

$$NQ (Triage.Queue) == 0.$$

As soon as this condition becomes true, the administrator entity proceeds to perform the action of dismissing the on-call doctor by entering the *Alter* module, labeled *Dismiss Extra Doctor*, whose dialog box is shown in Figure 5.39.

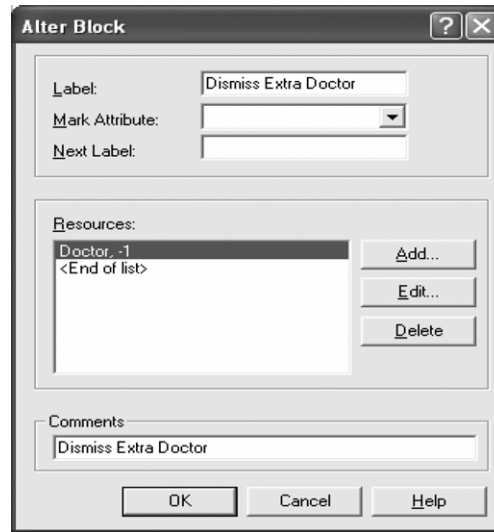


Figure 5.39 Dialog box of the *Alter* block *Dismiss Extra Doctor*.

Here the effect of the administrator entity is to reduce the capacity of the *Doctor* resource by 1 (note that a resource capacity cannot be reduced to a negative value).

Finally, the administrator entity loops back to the *Hold* module, called *Hold Until Triage Summons On-Call Doctor*, to start the next cycle of summoning/dismissing the on-call doctor. Thereafter, the administrator entity will continue traversing this loop indefinitely throughout a simulation run.

5.7.5 STATISTICS COLLECTION

In this model, statistics are collected using various methods as follows:

- Using *Record* modules
- Default collection of statistics by Arena
- Specifying statistics in the *Statistic* module

Statistics collection via *Record* modules was described in Section 5.4. Default collection of statistics is achieved in *Queue* and *Resource* spreadsheets by checking the last column (e.g., see Figure 5.31), and this is the default Arena setting. Finally, we illustrate statistics collection via the *Statistic* module, whose spreadsheet view is depicted in Figure 5.40.

Statistic - Advanced Process								
	Name	Type	Expression	Report Label	Frequency Type	Expression	Output File	Categories
1	Distribution of Doctors	Frequency		Distribution of	Value	NR(Doctor)		4 rows
2	Daily Throughput	Output	NC(Patient Departures)/365	Daily Throughput	Value	Expression 1		0 rows

Double-click here to add a new row.

Figure 5.40 Spreadsheet view of the *Statistic* module specifying frequency and output statistics.

This spreadsheet specifies collection of the distribution (*histogram*) of the number of doctors in the emergency room and the daily facility throughput.

5.7.6 SIMULATION OUTPUT

Figures 5.41 through 5.43 display reports of the results of a simulation run of length 525,600 minutes (1 year of emergency room operation).

Figure 5.41 displays statistics of patient sojourn time and patient flow through the emergency room. Here, the *Tally* section indicates that the tallied patient sojourn times, from patient arrival to patient discharge, last on average some 108 minutes, and the half-width of their 95% confidence interval is 2.3 minutes. However, the sojourn times have considerable variability as indicated by the minimal and maximal observed sojourn times. The *Counter* section records that the emergency room processed over 52,000 patients during its 1-year operation, while the *Output* section shows that the daily throughput was about 144 patients per day.

4:16:18PM

User Specified

September 28, 2005

Emergency Room

Replications: 1

Replication 1

Start Time: 0.00

Stop Time: 525,600.00

Time Units: Minutes

Tally

Interval	Average	Half Width	Minimum	Maximum
Patient Sojourn Time	108.07	2.31751	33.9229	510.04

Counter

Count	Value
Patient Departures	52,563.00

Output

Output	Value
Daily Throughput	144.01

Figure 5.41 Statistics of patient sojourn time and patient flow in the emergency room model.

Figure 5.42 displays utilization statistics of human resources in the emergency room, which consist of doctors, nurses, and receptionists. Recall that the numbers of nurses and receptionists at the emergency room are fixed throughout the simulation horizon, whereas the number of doctors is variable due to the periodic summoning and dismissal of the doctor on call.

4:15:21PM

Resources

September 28, 2005

Emergency Room

Replications: 1

Replication 1

Start Time:

0.00

Stop Time:

525,600.00

Time Units: Minutes

Resource Detail Summary

Usage

	<u>Inst Util</u>	<u>Num Busy</u>	<u>Num Sched</u>	<u>Num Seized</u>	<u>Sched Util</u>
Doctor	0.95	2.22	2.30	105,143.00	0.97
Nurse	0.26	0.51	2.00	31,521.00	0.26
Receptionist	0.80	2.40	3.00	105,148.00	0.80

3:09:27PM

Frequencies

November 30, 2005

Emergency Room

Replications: 1

Replication 1

Start Time:

0.00

Stop Time:

525,600.00

Time Units: Minutes

<u>Distribution of Doctors</u>	<u>Number Obs</u>	<u>Average Time</u>	<u>Standard Percent</u>	<u>Restricted Percent</u>
0 Doctor	1,971	5.7650	2.16	2.16
1 Doctor	7,012	4.2977	5.73	5.73
2 Doctors	8,904	35.4297	60.02	60.02
3 Doctors	3,863	43.6540	32.08	32.08

Figure 5.42 Statistics of human resource utilization in the emergency room model.

The *Usage* section in the *Resources* segment displays utilization-related statistics of emergency room (human) resources, taking into account the fact that the number of available resources (in this case doctors) may vary. These utilization statistics are computed over the simulation horizon as follows:

- The *Inst Util* column computes the time averages of instantaneous utilization. The *instantaneous utilization* of a resource is the fraction of busy resources to total available resources at any given time. For example, the instantaneous utilization of doctors is very high (95%), and would have been even higher had an on-call doctor not been available.
- The *Num Busy* column computes the time average of the number of busy resources.
- The *Num Sched* column computes the time average of the number of available resources.
- The *Num Seized* column computes the number of times a resource is seized.
- The *Sched Util* column computes the ratio of *Num Busy* to *Num Sched*.

The *Frequencies* segment displays distribution-related statistics of the random process of the number of busy doctors over time. These statistics are computed over the simulation horizon as follows:

- The *Distribution of Doctors* column lists all values (states) that can be assumed by the number of busy doctors.
- The *Number Obs* column tallies the observed frequency of each state listed.
- The *Average Time* column computes the average holding time in each state listed (i.e., average time spent in a state).
- The *Standard Percent* column computes the ratio of time spent in a state to the total simulation horizon and displays the ratio as a percentage. Note that these numbers provide an estimate of the probability distribution of the number of busy doctors.
- The *Restricted Percent* column is similar to the *Standard Percent* column except that some states may be excluded. Note that these numbers provide an estimate of the conditional probability distribution of the number of busy doctors, given that some states are excluded. Observe that in our case the two columns are identical since no exclusion was specified (in the *Statistic* module spreadsheet).

An examination of the *Frequencies* table clearly shows that the emergency room doctors are severely overworked! Indeed, this observation is consistent with the high doctor utilization in the *Usage* section.

Figure 5.43 displays waiting line statistics in the emergency room in terms of average waiting times and average number of patients in lines.

4:15:50PM

Queues

September 28, 2005

Emergency Room

Replications: 1

Replication 1

Start Time: 0.00 Stop Time: 525,600.00 Time Units: Minutes

Queue Detail Summary

Time

	<u>Waiting Time</u>
Check out.Queue	7.35
Hold Until Triage Dismisses On Call Doctor.Queue	40.46
Hold Until Triage Summons on call Doctor.Queue	95.57
Reception.Queue	7.49
Triage.Queue	17.16
Wait for Doctor.Queue	24.50
Waiting Room.Queue	0.48

Other

	<u>Number Waiting</u>
Check out.Queue	0.73
Hold Until Triage Dismisses On Call Doctor.Queue	0.30
Hold Until Triage Summons on call Doctor.Queue	0.70
Reception.Queue	0.75
Triage.Queue	1.72
Wait for Doctor.Queue	2.45
Waiting Room.Queue	0.03

Figure 5.43 Waiting lines statistics in the emergency room model.

These include two types of statistics:

- Patient waiting times at various stages of their sojourn in the emergency room.
- The on-call doctor's state (on duty and off duty), which are computed from waiting times of the administrator entity.

The *Time* section shows that the averages of waiting times in the reception queue and the checkout queue are comparable and relatively significant at around 7 minutes. Indeed the utilization of the receptionists is quite high at 80% (see Figure 5.42). However, the corresponding averages of patients waiting in those lines, shown in the *Other* section, are reasonable (under one person on average). In a similar vein, average patient waiting times in the triage queue and for treatment by a doctor are high (17 minutes and 25 minutes, respectively), and lead to higher average numbers of waiting patients in those lines. This is an expected consequence of the fact that doctors are overworked. By contrast, average waiting times for nurses in the waiting room is very low (less than a minute), as is the average number of such patients (just 0.03). This fact is borne out by the low utilization of nurses (26%) in Figure 5.42. Finally, the averages of times on duty and off duty of the on-call doctor show that the former is about half the latter. Thus, the on-call doctor is not heavily utilized.

5.8 SPECIFYING TIME-DEPENDENT PARAMETERS VIA A SCHEDULE

Our examples have so far assumed that random phenomena (e.g., arrivals, services, etc.) are modeled as variates from a fixed probability law that does not change in time, so that the underlying process is *stationary* (see Section 3.9). However, it is quite common in practice for the underlying probability law to vary in time, in which case the process is *nonstationary* (time dependent). For example, researchers debate whether recent indications of rising global temperatures are ordinary fluctuations of a stationary temperature process or an indication of a change in the underlying probability law corresponding to a nonstationary temperature process (global warming). On the other hand, many types of arrival events (e.g., customer arrivals in stores, banks, or factories) are known to be nonstationary. Typical examples include the “rush hour” phenomenon (temporary heavy traffic) or the “ebb hour” phenomenon (temporary light traffic). Thus, a bank operation may experience “rush hour” periods in the morning (customers stopping by on their way to work) and at lunch time (customers using part of their lunch time for banking), while mid-morning hours may become “ebb hour” periods. This pattern may recur day after day, with some random fluctuations. A common special case of nonstationarity is when the parameters of the underlying probability law change in time. For example, the arrival rate of a Poisson process may change in the course of a day, season, and so on, in accordance with “rush hour” or “ebb hour” phenomena. This section discusses Arena facilities that permit the modeling of time-dependent parameters of random processes (e.g., arrivals, service, and resource capacities), by varying such parameters over time via a schedule specification.

We first illustrate this facility by a time-dependent arrival specification. The Arena *Create* module provides a *Schedule* option in its *Time Between Arrivals* section, but only for exponential interarrival times, as depicted in Figure 5.44.

The *Schedule Name* field specifies that entity generation is governed by a *Schedule* module called *Schedule 1*. The spreadsheet view of the *Schedule 1* module is displayed in Figure 5.45.

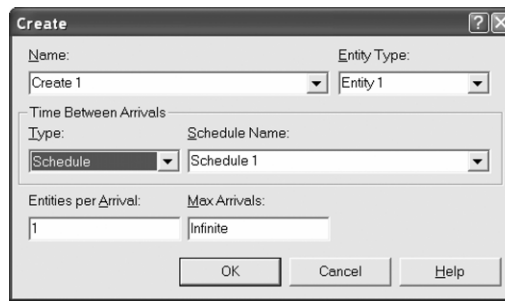


Figure 5.44 Dialog box of a *Create* module with the *Schedule* option.

Schedule - Basic Process						
	Name	Format Type	Type	Time Units	Scale Factor	Durations
1	Schedule 1	Duration	Arrival	Hours	1.0	24 rows

Figure 5.45 Spreadsheet view of the *Schedule* module *Schedule 1*.

The *Arrival* option is selected in the *Type* column to indicate that the schedule is dedicated to an arrival process. The *Format Type* column declares whether the schedule is specified by durations or by a calendar, and the *Scale Factor* column may be used to scale all schedule magnitudes (the default is 1.0, meaning no scaling). Under the *Durations* heading is a button labeled *24 rows*, which specifies a time span consisting of 24 consecutive time slots (intervals). Clicking this button pops up a dialog box, depicted in Figure 5.46.

The graph of Figure 5.46 specifies the hourly arrival rates over a time span of 24 hours, depicted as bars over 24 time slots; these correspond to the 24 rows (or durations) alluded to in Figure 5.45. More specifically, the horizontal axis is a time axis divided into consecutive 1-hour time slots (intervals), while the vertical axis is a magnitude axis for the associated arrival rates. Using mouse clicks at appropriate points within the bars of the graph, the modeler can visually specify the requisite hourly arrival rates.

Note carefully that Arena selects the requisite arrival rate only when the most recent arrival “shows up” in the model. This means that if no arrival fell within a given time slot, then no arrivals would be generated with that slot’s arrival rate! In other words, if an interarrival interval wholly contains a slot, then no arrivals from such a slot will be scheduled. This might create some modeling problems when the arrival rates vary significantly from slot to slot.

The modeler may interpret the actual time at will, since the first time slot (labeled by *Day 1 00:00:00*) is just a convention for the time origin. In our example, the first time slot in Figure 5.45 actually corresponds to the time interval 8:00 A.M. to 9:00 A.M.—a morning “rush hour” period with a high traffic rate. The hourly rate progressively ebbs towards midday, and it then picks up gradually in the afternoon, reaching its peak in the evening “rush hour” of 5:00 P.M. to 6:00 P.M. It then ebbs again during the night, bottoming out at 12:00 A.M. Finally, it rises again towards the morning “rush hour” of the next day (*Day 2 00:00:00*). This pattern of time-dependent arrival rates repeats on each subsequent day.

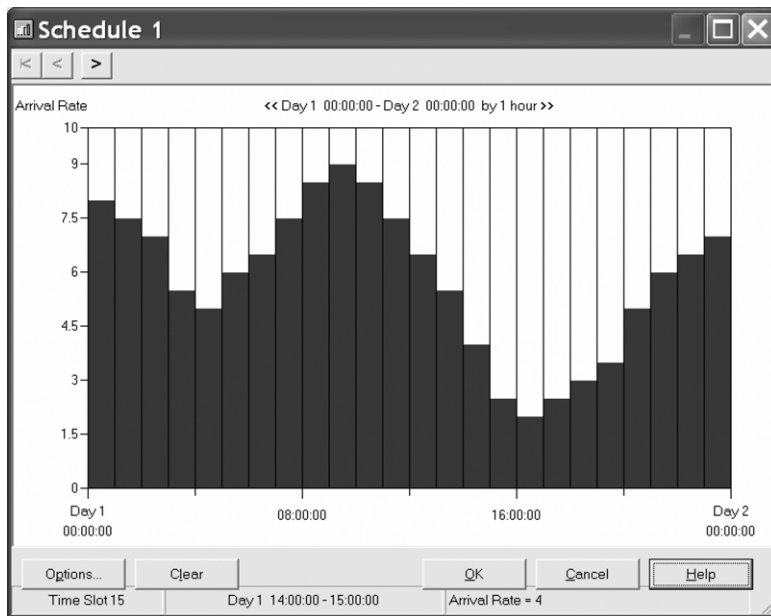


Figure 5.46 Dialog box for schedule *Schedule 1* specifying time-dependent arrival rates.

To customize the *Schedule* module, the modeler clicks the *Options...* button in Figure 5.46 to pop up the dialog box displayed in Figure 5.47. Here, the *X-axis* section has a *Time slot duration* field to specify the time unit (time slot width) on the horizontal axis, and a *Range* field to specify the number of time slots in the graph. The *Calendar speed* field is a user interface parameter that specifies the scrolling speed of the graph (using the scrolling buttons below the title bar in Figure 5.46), when the graph does not fit into its canvas. The *Y-axis* section has *Maximum* and

Section	Field	Value
X-axis	Time slot duration	1 hour
	Range (time slots)	24
	Calendar speed (slots per click)	5
	Show vertical grid lines	<input checked="" type="checkbox"/>
Y-axis	Maximum	10
	Minimum	0
	Snap spacing	1
When at end of schedule	Repeat from beginning	<input checked="" type="radio"/>
	Remain at capacity	<input type="radio"/> 0

Figure 5.47 Dialog box of the *Options...* field of schedule *Schedule 1*.

Minimum field values for the vertical axis. The *Snap Spacing* field is a user interface parameter to specify the granularity of arrival rates (on the vertical axis) via button clicks as described above.

The replication length may precisely equal the time span of the graph in Figure 5.46, but in the majority of cases it is either shorter or longer than the specified time span. If the replication length is shorter, then the rest of the graph is simply ignored. Otherwise, the information in the *When at end of schedule* section specifies how to assign arrival rates beyond the time span of the schedule. Two options (radio buttons) are provided:

- The *Repeat from beginning* option simply cycles back and repeats the schedule from its beginning for each time span. In our example, the schedule in Figure 5.46 is a daily schedule, to be repeated for each simulated day.
- The *Remain at capacity* option provides a data field specifying a fixed arrival rate. The replication will follow the schedule for its first time span. However, once the simulation clock exceeds the time span, that fixed arrival rate will be in effect for the rest of the replication.

In a similar vein, the user can schedule time-dependent resource capacities. To this end, the user selects the *Based on Schedule* option in the spreadsheet view of the *Resource* module, and then proceeds to specify a resource capacity schedule analogously to the arrivals example above. What happens when resource capacity is decreased while the resource is in use? To handle such eventualities, Arena offers users three options in the *Schedule Rule* column of the *Resource* spreadsheet view when the *Based on Schedule* option is selected:

- The *Ignore* option starts the time duration of the schedule change immediately, but each “excess” resource is removed only after it is released by the entity currently seizing it.
- The *Wait* option waits to start the time duration of the schedule change until the last “excess” resource is released by the entity currently seizing it.
- The *Preempt* option starts the time duration of the schedule change immediately, but “interrupts” processing by promptly releasing “excess” resources and returning the seizing entities to their queues.

Incidentally, machine failures in Arena are implemented using these same options (see Section 11.7). Finally, a schedule specification can be similarly used in Arena to vary the value of any parameter over time by selecting the *Other* option in the *Type* column of the spreadsheet view of the *Schedule* module.

EXERCISES

1. *Press operation.* The press department of an automobile manufacturing facility runs two main operations, each with its own press machine: *front-plate press* operation and *rear-plate press* operation. These operations can be performed in any order, but both have to be performed for each arriving plate. Plates (jobs) arrive randomly and their interarrival times are exponentially distributed with mean 5 minutes. The service time in the *front-plate press* operation is distributed iid Unif(1, 5) minutes, and in the *rear-plate press* operation it is distributed iid Unif(2, 6) minutes. A plate joins the queue of the press operation with the least number of plates waiting at that time (since there is no sequencing requirement),

and on completion joins the queue of the other press operation after which it departs from the system. Finally, the press department is a three-shift facility running 24 hours a day.

- a. Develop an Arena model of the press department, and simulate it for one year.
 - b. Estimate the following statistics:
 - Average time arriving plates spend in the press department
 - Utilization of the press machine in each operation
 - Average queue delay at each operation
 - Average time in the press department of those arriving plates that join first the *rear-plate press* operation, and then proceed to the *front-plate press* operation
2. *Electrolytic forming process.* An expensive custom-built product goes through two stages of operation. The first stage is an *electrolytic forming* process, served by two independently operating forming machines, where the product is built in a chemical operation that must conform to precise specifications. The second stage is a *plating operation* in which the product is silver plated. Customer orders arrive with interarrival times distributed iid $\text{Tria}(3, 7, 14)$ hours, and join a queue in front of the forming process. The electrolytic forming processing time is distributed iid $\text{Unif}(8, 12)$ hours. The silver-plating process also has a queue in front of it. Plating time is distributed iid $\text{Unif}(4, 8)$ hours. The variability in the processing times is due to design variations of the incoming orders.

The two processes do not perform perfectly. In fact, 15% of the jobs that emerge from the forming process and 12% of the jobs that emerge from the plating process are defective and have to be reworked. All defective jobs are sent to a single rework facility, where design modifications and corrections are performed manually. However, plating reworks have a lower priority than forming modifications. Plating rework times are distributed iid $\text{Unif}(15, 24)$ hours, while forming reworks are distributed iid $\text{Unif}(10, 20)$ hours. Jobs departing from the rework facility go back to the process they came from to redo the operation found defective. Jobs that successfully complete the plating process leave the facility. Note that a job may go back and forth between a process and the rework operation any number of times.

- a. Develop an Arena model of the electrolytic forming process, and simulate it for 1 year (24 hours of continuous operation).
 - b. How busy are each of the two operations and the rework facility?
 - c. What are the expected delays in process queues and the rework facility?
 - d. What is the expected job flow time throughout the entire facility?
 - e. Suggest a change in the system to reduce (even slightly) the expected job flow time. Run the modified model and compare the job flow statistics.
3. *Supermarket cashier management.* A supermarket is open 24/7 and operates in 3 shifts: first shift from 8:00 A.M. to 4:00 P.M., second shift from 4:00 P.M. to 12:00 A.M., and third shift from 12:00 A.M. to 8:00 A.M. Customers arrive according to a Poisson process with shift-dependent arrival rates, and their shopping times (excluding checkout) are iid but shift dependent. Consequently, the supermarket management assigns variable numbers of cashiers per shift. The arrival, shopping, and cashier parameters are displayed in the following table.

After shopping, customers queue up in a single line for checkout. Checkout times of customers are iid, but shopping-time dependent as follows: a customer's checkout

Shift Number	Arrival Rate (customers per minute)	Shopping Time Distribution (minutes)	Number of Cashiers
1	16.8	Unif(5, 15)	2
2	24.0	Unif(15, 40)	4
3	0.7	Unif(1, 5)	1

time is 2 minutes plus a random fraction of its shopping time, where the fraction is iid triangular between 20% and 30%, with a most likely value of 25%. Finally, a customer is assigned to the shift during which it departs from the supermarket (note that a customer may arrive during one shift and depart during a subsequent one).

- Develop an Arena model of the supermarket, and simulate it for 30 days.
- Compute the average, variance, and squared coefficient of variation of customer waiting times for a cashier (excluding checkout processing times).
- What is the per-shift average sojourn time of customers in the supermarket?
- What are the overall instant and scheduled cashier utilizations?