Chapter 5

Arena Basics
The Arena Simulation System

- **Arena** is a powerful simulation environment
  - consists of modeling object templates, called *modules*, and transactions that move among them, called *entities*
  - has a *visual* front-end
  - built around SIMAN block-oriented language constructs and other facilities

- **SIMAN** consists of two classes of objects:
  - **Blocks** are basic logic constructs that represent operations, such as *SEIZE* blocks that model seizing of a facility by a *transaction* entity, while *RELEASE* blocks release the facility for use by other *transaction* entities
  - **Elements** are objects that represent facilities, such as *RESOURCES* and *QUEUES*

- Arena modules are selected from template panels
  - examples: *Basic Process, Advanced Process, Advanced Transfer*

- Arena modules are high-level constructs that functionally equivalent to sets of SIMAN blocks and/or elements, and internally are built of SIMAN blocks and/or elements
The Arena Home Screen

- Title Bar
- Menu Bar
- Run Interaction Toolbar
- View Toolbar
- Standard Toolbar
- Project Bar Toolbar
- Drawing Toolbar
- Animate Toolbar
- Template Panel
- Animate Transfer Toolbar
- Model Window Canvas
  - Flowchart View
- Model Window Canvas
  - Spreadsheet View
Example: A Simple Workstation

• Consider a single workstation, known in queueing theory as the M/M/1 queue, where
  • there is a machine with an infinite buffer in front of it
  • jobs arrive randomly and wait in the buffer while the machine is busy
  • jobs are processed by the machine and then leave the system
  • job inter-arrival times are exponentially distributed with mean 30 minutes
  • job processing times are exponentially distributed with mean 24 minutes

![Diagram of Create 1, Process 1, Dispose 1 processes]
Simulation Objects and Actions

• Simulating the above workstation calls for the following actions:
  • jobs are created, one at a time, according to their arrival distribution
  • if the machine is busy processing another job, then the arriving job is queued in the buffer
  • 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 its processing-time distribution
  • on process completion, the job departs the machine and is removed from the system

• Simulation objects and their actions and interactions
  • are modeled by Arena modules
  • parameterized by associated dialog boxes
Create Module

- The *Create* module generates a stream of arrivals of Arena entities (jobs, people, messages, etc.)

![Dialog box for a Create module](image)

**Dialog box for a Create module**
Create Module (Cont.)

• The Type pull-down menu for the Time Between Arrivals field offers the following options:
  • Random (exponential inter-arrival 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 inter arrival times)
  • Expression (any type of inter-arrival time pattern specified by an Arena expression, including Arena distributions)
**Process Module**

- The *Process* module processes (serves) Arena *entities*
• 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
  • the processing (holding) time of a resource (called *Machine* in the example) by an *entity* is specified via a **DELAY** block within the *Process* module
Dispose Module

- The *Dispose* module implements an entity “sunset” mechanism
  - *entities* that enter it are simply discarded
Whenever an Arena model is saved, the model is placed in a file with a .doe extension (e.g., mymodel.doe).

Whenever an Arena model (say, 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:

- mymodel.p (program file)
- mymodel.mdb (Access database file)
- mymodel.err (errors file)
- mymodel.opw (model components file)
- mymodel.out (SIMAN output report file)
The end-result of a simulation run is a set of requisite statistics, referred to as **run results**, such as:

- mean waiting times
- buffer content probabilities
- resource utilization

Arena provides a considerable number of default statistics in a report that is automatically generated at the end of a simulation run in Arena **reports**.

Additional statistics can be obtained by adding statistics collection modules in the model, such as:

- **Record** *(Basic Process template panel)*
- **Statistic** *(Advanced Process template panel)*
Example: a **Resources** Report

<table>
<thead>
<tr>
<th>Usage</th>
<th>Average</th>
<th>Half Width</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number Busy</td>
<td>0.7976</td>
<td>0.016547523</td>
<td>0</td>
<td>1.0000</td>
</tr>
<tr>
<td>Number Scheduled</td>
<td>1.0000</td>
<td>(Insufficient)</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>Utilization</td>
<td>0.7976</td>
<td>0.016547523</td>
<td>0</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number Times Used</td>
<td>19,890.00</td>
</tr>
<tr>
<td>Scheduled Utilization</td>
<td>0.7976</td>
</tr>
</tbody>
</table>

**Resources statistics from a single replication of the simple workstation model**
Example: a *Queues* Report

<table>
<thead>
<tr>
<th>A Work Station</th>
<th>Replications: 1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Replication 1</th>
<th>Start Time: 0.00</th>
<th>Stop Time: 600,000.00</th>
<th>Time Units: Minutes</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Process 1.Queue</th>
<th>Time</th>
<th>Average</th>
<th>Half Width</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Waiting Time</td>
<td>96.3230</td>
<td>14.85368</td>
<td>0</td>
<td>947.77</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>3.1937</td>
<td>0.532298979</td>
<td>0</td>
<td>42.0000</td>
</tr>
</tbody>
</table>

*Queues* statistics from a single replication of the simple workstation model
Arena Data Storage Objects

• Arena variables are user-defined global data storage objects used to store and modify state information either 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

• Arena Attributes are data storage objects associated with entities
  • unlike variables, attributes are local to entities in the sense that each instance of an entity has its own copy of the attributes

• Arena expressions can be viewed as specialized variables that store the value of an associated formula (expression)
Arena Statistics Collection

• Statistics collection via the **Statistic** module
  • **Time-Persistent** statistics are time-average statistics (e.g., average queue lengths, server utilization and various probabilities), and this option can be used to estimate any user-defined probability or time average
  • **Tally** statistics are customer averages, and have to be specified in a **Record** module to initiate statistics collection, but it is advisable to include their definition in the Statistic module as well, so the entire set of statistics may be viewed in the same spreadsheet for modeling convenience
  • **Counter** statistics are used to keep track of counts, and have to be specified in a **Record** module to initiate statistics collection
  • **Output** statistics are obtained by evaluating an expression at the end of a simulation run, using Arena variables, such as \( \text{DAVG}(S) \) (the time average of the time-persistent statistic \( S \)), \( \text{TAVG}(S) \) (the average of tally element \( S \)), \( \text{TFIN} \) (simulation completion time), etc.
  • **Frequency** statistics are used to produce frequency distributions of (random) expressions, such as Arena variables or resource states, allowing the estimation of steady-state probabilities

• Statistics collection via the **Record** module is achieved by its proper placement in the model
Record Module

- The **Record** module is used by entities to collect statistics at selected locations in the model
  - the dialog box below the list of statistics types in a **Record** module

![Dialog box for a Record module]
Record Module (Cont.)

- The **Record** module has the following options in its **Type** field:
  - the **Count** option maintains a count with a prescribed increment (any real value, positive or negative), which may be defined as any expression or function, and the counter is incremented whenever an **entity** enters the **Record** module.
  - the **Entity Statistics** option provides information on **entities**, such as time and costing/duration information.
  - the **Time Interval** option tallies the difference between the current time and the time stored in a prescribed attribute of the entering entity.
  - the **Time Between** option tallies the time interval between consecutive entries of entities in the **Record** module (these intervals correspond to inter-departure times from the module, and the reciprocal of the mean inter-departure times is the module’s throughput).
  - the **Expression** option tallies an expression, whose value is recomputed whenever an **entity** enters the **Record** module.
**Arena Output Reports**

- Arena output reports consist of two types: **automatic** and **user-specified**
- An **automatic** report of summary statistics is generated automatically at the end of a simulation run by a number of Arena constructs, such as **entities**, **queues** and **resources**
  - those statistics are implicitly specified by the modeler simply by dragging and dropping those modules into an Arena model
  - no further action is required of the user
- A **user-specified** report provides additional statistics via the **Statistic** module (**Advanced Process** template panel) and the **Record** module (**Basic Process** template panel)
  - obtained by explicitly specifying statistics collection in those modules, where the **Statistic** module is specified in a spreadsheet view, and the **Record** module must be placed in the appropriate location in the model
Example: Two Processes in Series

- Consider a **manufacturing network** of two workstations in series, consisting of an assembly workstation followed by a painting workstation, where
  - jobs arrive at the assembly station with exponentially distributed inter-arrival times of mean 5 hours
  - the assembly process always has all the raw materials necessary to carry out the assembly operation
  - the assembly time is uniformly distributed between 2 and 6 hours
  - after the process is completed, a quality control test is performed, and past data reveal that 15% of the jobs fail the test and go back to the assembly operation for rework
  - jobs that pass the test proceed to the painting operation that takes 3 hours for each unit

- We are interested in
  - simulating the system for 100,000 hours
  - estimating process utilizations, average job waiting times and average job flow times (the elapsed time for a job from start to finish).
The Arena Model

Two Processes in Series

Arena manufacturing model of assembly and painting processes in series
Assign Module

- The Assign module is used by entities to assign values to attributes
- The dialog box below assigns an arrival time to a job attribute

Dialog boxes for an Assign module
Decide Module

- The *Decide* module is used by *entities* to make branching decisions, based on chance or the truth/falsity of prescribed conditions
  - the dialog box below makes a two-way probabilistic branching decision

![Dialog box for a Decide module](image-url)
**Record Module**

- The *Record* module is used by *entities* to collect statistics
  - the dialog box below tallies job flow times

**Dialog box for a Record module**
# Resources Report

**Assembly Op.**

<table>
<thead>
<tr>
<th>Replication 1</th>
<th>Start Time:</th>
<th>0.00</th>
<th>Stop Time:</th>
<th>100,000.00</th>
<th>Time Units:</th>
<th>Hours</th>
</tr>
</thead>
</table>

## Resource Detail Summary

### Usage

<table>
<thead>
<tr>
<th>Resource</th>
<th>Number Busy</th>
<th>Number Scheduled</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSEMBLER</td>
<td>0.59</td>
<td>1.00</td>
<td>0.59</td>
</tr>
<tr>
<td>PAINTER</td>
<td>0.38</td>
<td>1.00</td>
<td>0.38</td>
</tr>
</tbody>
</table>

### Other

<table>
<thead>
<tr>
<th>Resource</th>
<th>Number Times Used</th>
<th>Scheduled Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSEMBLER</td>
<td>14741.00</td>
<td>0.59</td>
</tr>
<tr>
<td>PAINTER</td>
<td>12525.00</td>
<td>0.38</td>
</tr>
</tbody>
</table>
### Queue Report

**Assembly Op.**

<table>
<thead>
<tr>
<th>Replication 1</th>
<th>Start Time: 0.00</th>
<th>Stop Time: 100,000.00</th>
<th>Time Units: Hours</th>
</tr>
</thead>
</table>

#### Queue Detail Summary

**Time**

<table>
<thead>
<tr>
<th>Queue Type</th>
<th>Waiting Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly.Queue</td>
<td>3.49</td>
</tr>
<tr>
<td>Painting.Queue</td>
<td>0.08</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>3.57</td>
</tr>
</tbody>
</table>

**Other**

<table>
<thead>
<tr>
<th>Queue Type</th>
<th>Number Waiting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly.Queue</td>
<td>0.51</td>
</tr>
<tr>
<td>Painting.Queue</td>
<td>0.01</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>0.53</td>
</tr>
</tbody>
</table>
User Specified Report

User Specified

Assembly Op.  Replications: 1

Replication 1  Start Time: 0.00  Stop Time: 100,000.00  Time Units: Hours

Tally

<table>
<thead>
<tr>
<th>Interval</th>
<th>Average</th>
<th>Half Width</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow Time</td>
<td>11.8873</td>
<td>0.495720704</td>
<td>5.0000</td>
<td>130.42</td>
</tr>
</tbody>
</table>
Example: Production/Inventory System

- Consider a production/inventory system in which
  - the production process (e.g., packaging) is comprised of three stages
    1. filling each container unit (e.g., bottles)
    2. sealing each unit
    3. placing labels on each unit
  - a raw-material storage feeds the production process, and finished units are stored in a warehouse
  - customers arrive at the warehouse with product requests (demands), and if a request cannot be fully satisfied by on-hand inventory, the unsatisfied portion represents lost business
The Production Operation

• The production process component operates as follows:
  • there is always sufficient raw material in storage, so the production process never starves
  • product processing is carried out in lots of 5 units, and finished lots are placed in the warehouse
  • lot processing time is uniformly distributed between 10 and 20 minutes
  • the production process experiences random failures, which may occur at any point in time
  • times between failures are exponentially distributed with a mean of 200 minutes
  • repair times are normally distributed, with a mean of 90 minutes and a standard deviation of 45 minutes
The Warehouse Operation

• The warehouse component operates as follows:
  • warehouse operations implement the \((r, R)\) inventory control policy, where
    1. the warehouse has a **capacity** (target level) of \(R = 500\) units
    2. the production process stops when the inventory in the warehouse reaches the target level
    3. from this point and on, the production process remains inactive until the inventory level drops to or below the reorder point of \(r = 150\) units
    4. At this point the production process is restarted until the warehouse inventory level reaches the target level \(R = 500\)
  • the inter-arrival times between successive customers are uniformly distributed between 3 to 7 hours, and individual demand sizes are distributed uniformly between 50 and 100 units
  • on customer arrival, the inventory is immediately checked,
    • and if there is sufficient stock on hand, that demand is promptly satisfied,
    • and otherwise, the unsatisfied portion of the demand is lost
System Performance Measures

- The performance measures of interest are:
  - process utilization
  - process downtime probability
  - warehouse average inventory level
  - percentage of customers whose demand is not completely satisfied on arrival at the warehouse
  - average size of lost customer demands at the warehouse, given that the demands are not completely satisfied
Production/Inventory Arena Model

[Diagram showing the Production/Inventory Arena model with various processes and decision points such as Create, Seize, Hold, Release, Check Target, Update Inventory, Demand Management, and Inventory Management.]
Arena Model Logic

• The Arena model of the production/inventory system is composed of two segments: **inventory management** and **demand management**

• In the Arena model logic of the **inventory management segment**:
  • the packaging process takes a unit of raw material from its queue, and processes it as a batch of 5
  • the finished lot is added to the warehouse inventory (variable *Inventory*)
  • if *Inventory* reaches or up-crosses the target level (variable *Target Stock*), then production stops until the reorder point (variable *Reorder Point*) is reached or down-crossed again
  • processing of a new batch starts immediately when the reorder point is down-crossed

• In the Arena model logic of the **demand management segment**:
  • arrivals of customers and their demands at the warehouse are generated
  • the variable *Inventory* is adjusted upon customer arrival
  • the value of *Inventory* is monitored for triggering resumption of suspended production when the reorder point is down-crossed
  • track is kept of lost demand
Inventory Management Logic

• The Arena model logic implementation of the inventory management segment makes use of the following modules:
  • the Create module, called Raw Material, generates product units for the packaging (batching) operation
  • the packaging operation is modeled using a sequence of Seize, Delay and Release modules
  • the actual processing (packaging in our case) takes place at the Delay module, called Packaging Process, where the packaging time of a batch is specified as $\text{Unif}(10,20)$ minutes
**Seize and Release Modules**

- The *Seize* module is used by *entities* to seize resources
- The *Release* module is used by *entities* to release resources
  - the dialog boxes below seize and release a packaging machine

---

**Dialog boxes for a Seize module and a Release module**
Resource Module

• A spreadsheet view of Resource modules (from the Basic Process template panel) is used to list all resources and their attributes
• One of the attribute fields is Failures, which specifies resource failures in a dialog box
  • the dialog box below shows the resource Packaging Process with its Failures field details

Spreadsheet views for a Resource module (bottom) and its Failures dialog box (top)
**Failure Module**

- All failure/repair data (uptimes and downtimes) are specified in the spreadsheet *Failure* module (from the *Advanced Process* template panel)
  - the dialog box below illustrates the data for a random failure

![Dialog box of a Failure module](image-url)
**Hold Module**

- The *Hold* module performs a gating function on *entities*
  - the dialog box below illustrates the use of a *Hold* module for checking the production state

![Dialog Box of a Hold module](image)
Hold Module (Cont.)

• The state of production (0 = off or 1 = on) is maintained in a variable, called Production, initially set to 0
  • the Hold module scan for the truth or falsehood of a logical condition, (Production = 1), and holds the entity in the module until the condition becomes true
  • once the condition becomes true, the entity proceeds to the next module, and otherwise, it waits on in a Queue, called Shall We Produce?.Queue

• The inventory level at the warehouse is maintained in a variable, called Inventory, initially set to 250
  • each product entity (batch of 5 units) that enters the Assign module, called Update Inventory, increases the warehouse inventory level by a batch of 5 finished jobs by adding the Batch Size to the variable Inventory
  • that entity then proceeds from the Hold module, called Shall We Produce?, to the Decide module, called Check Target
**Decide Module**

- The *Decide* module is used by *entities* to make branching decisions, based on chance or the truth/falsity of prescribed conditions.
  - The dialog box below makes a two-way probabilistic branching decision by checking the inventory target level.
**Decide Module (Cont.)**

- The *entity* tests whether or not the inventory target level has been up-crossed as follows:
  - if the inventory target level has been up-crossed, then the *entity* moves on to the *Assign* module, called *Stop Production*, and sets *Production* = 0 to signal that production is suspended
  - otherwise, the *entity* does nothing
Wrapping Up *Entity* Sojourns

• An *entity* has completed its sojourn through the system, and would normally be disposed of (at a *Dispose* module)

• However, since the *Packaging* module is never starved, and there are no delays incurred since its departure from the *Packaging* station, we can “recycle” the *entity* by always sending it back to the *Packaging Queue* to play the role of a new arrival

• This modeling device is logically equivalent to disposing of the *entity* and creating a new
  • however, it is computationally more efficient, since it saves us this extra computational effort
  • thus, the simulation will run faster!
Demand Management Logic

• The Arena model logic implementation of the demand management segment makes use of the following modules:
  
  • the Create module, called Customer Arrives, generates arrivals of customers, and the Assign module, called Customer Demand, assigns to the customer a demand level
  
  • the \((r,R)\) inventory policy is modeled using a sequence of Decide and Assign modules
  
  • the Inventory variable keeps track of the inventory level in the warehouse
Create Module

• The Create module generates a stream of arrivals of Arena entities (jobs, people, demands, etc.)
  • the dialog box below models the arrival of customer demands

Dialog Box of a Create module
Create and Assign Modules

• The arrival pattern of customers is specified to be random with inter-arrival time distribution \( \text{Unif}(3,7) \)

• On arrival, the customer entity first enters the Assign module, called Customer Demand, where its Demand attribute is assigned a random value from the distribution \( \text{Unif}(50,100) \)
The customer then proceeds to the **Decide** module, called **Check Inventory**, to test whether the warehouse has sufficient inventory on hand to satisfy its demand.

If the variable **Inventory** has an equal or larger value than attribute **Demand**, then:

- the customer takes the **True** exit to the **Assign** module, called **Take Away From Inventory**, where its inventory is decreased by the demand amount.
- it next proceeds to the **Decide** module, named **Restart Production**, to test whether the **Reorder Level** variable has just been down crossed.
- if it was down crossed, the customer proceeds to the **Assign** module, called **Production Start**, to set **Production = 1**, which would promptly release the product entity currently detained in the **Hold** module **Shall We Produce?**, effectively resuming the production process.
- either way, the Customer entity proceeds to be disposed of at the **Dispose** module, called **Dispose1**.
Decide Module (Cont.)

• If the value of variable *Inventory* is smaller than attribute *Demand*, then the current demand is either partially satisfied or not at all

  • either way, the customer *entity* proceeds to the *Assign* module, called *Lost Customer*, where it sets the *Inventory* variable to 0
  • there it updates the variable *Lost*, which keeps track of customers whose demand could not be fully satisfied (*Lost = Lost + 1*), and the variable *Lost Amount*, which keeps track of the customer’s demand lost (*Lost Amount = Demand – Inventory*)
  • the customer *entity* next enters the *Record* module, called *Tally Amount Lost*, to tally the lost quantity per customer whose demand was not fully satisfied
  • the customer *entity* then proceeds to be disposed of at the module called *Dispose1*
**Variable Module**

- The *Variable* spreadsheet module (*Basic Process* template panel) is used to set or inspected user-defined variables defined in the model and their properties
  - the dialog box bellow lists user defined variables for our example system

<table>
<thead>
<tr>
<th>Variable - Basic Process</th>
<th>Name</th>
<th>Rows</th>
<th>Columns</th>
<th>Clear Option</th>
<th>Initial Values</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Inventory</td>
<td></td>
<td>System</td>
<td></td>
<td>1 rows</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Batch Size</td>
<td></td>
<td>System</td>
<td></td>
<td>1 rows</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Target Stock</td>
<td></td>
<td>System</td>
<td></td>
<td>1 rows</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Production</td>
<td></td>
<td>System</td>
<td></td>
<td>1 rows</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Reorder Point</td>
<td></td>
<td>System</td>
<td></td>
<td>1 rows</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Demand</td>
<td></td>
<td>System</td>
<td></td>
<td>0 rows</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Lost</td>
<td></td>
<td>System</td>
<td></td>
<td>0 rows</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Total Customers</td>
<td></td>
<td>System</td>
<td></td>
<td>0 rows</td>
<td></td>
</tr>
</tbody>
</table>

**Dialog Box of a Variable module**
**Statistic Module**

- The *Statistic* spreadsheet module (*Advanced Process* template panel) is used to specify statistics collection in the model and their properties
  - the dialog box below lists statistics collected in our example system

![Dialog Box of a Statistic module](image-url)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Expression</th>
<th>Report Label</th>
<th>Frequency Type</th>
<th>Resource Name</th>
<th>Report Label</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stock on Hand</td>
<td>Time-Persistent</td>
<td>Inventory</td>
<td>Stock on Hand</td>
<td>Value</td>
<td>Stock on Hand</td>
<td>0 rows</td>
<td></td>
</tr>
<tr>
<td>Process States</td>
<td>Frequency</td>
<td></td>
<td>Process States</td>
<td>State</td>
<td>Packaging Process</td>
<td>Process States</td>
<td>0 rows</td>
</tr>
<tr>
<td>Production On</td>
<td>Time-Persistent</td>
<td>Production==1</td>
<td>Production On</td>
<td>Value</td>
<td>Production On</td>
<td>0 rows</td>
<td></td>
</tr>
<tr>
<td>Lost Percentage</td>
<td>Output</td>
<td>Lost/Total Customers</td>
<td>Lost Percentage</td>
<td>Value</td>
<td>Lost Percentage</td>
<td>0 rows</td>
<td></td>
</tr>
</tbody>
</table>
Statistics listed include
- a *Time-Persistent* statistic, called *Stock On Hand*, for the *Inventory* variable
- a *Time-Persistent* statistic, called *Production On*, for the condition *Production = 1*
- a *Frequency* statistic, called *Process States*, which estimates the state probabilities of the packaging process, namely, the probabilities that the packaging process is busy or down

The statistical outputs for *Inventory* include
- the average value
- 95% confidence interval
- minimal and maximal values

The statistical output for state *Production = 1* is the percentage of time this expression is true, that is, the probability that the packaging process is in production
Record Module

• The Record module is used to specify user-defined statistics collection via entities in particular model locations
  • the dialog box below specifies the tallying of demand loss at a Record module, called Tally Lost Amount

![Dialog Box of a Record module]
\textit{Record Module (Cont.)}

- Whenever a customer \textit{entity} enters the module, the expression (in this case, the variable) \textit{Amount Lost} is evaluated, and the resultant value is tallied.

- When the replication terminates, the output report will contain a \textit{Tallies} section summarizing the statistics of the amount lost per customer.
Simulation Output Reports

• The simulation was run for one replication of length 1,000,000 minutes (slightly less than two years)

• The reports produced include
  • *User Specified* report
  • *Frequencies* report
# User Specified Report

## Pack Process

**Replications:** 1

### Replication 1

<table>
<thead>
<tr>
<th>Start Time</th>
<th>Stop Time</th>
<th>Time Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>1,000,000.00</td>
<td>Minutes</td>
</tr>
</tbody>
</table>

### Tally

<table>
<thead>
<tr>
<th>Expression</th>
<th>Average</th>
<th>Half Width</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount Lost Per Customer</td>
<td>25.1504</td>
<td>1.48315</td>
<td>0.00985116</td>
<td>85.5513</td>
</tr>
</tbody>
</table>

### Time Persistent

<table>
<thead>
<tr>
<th>Time Persistent</th>
<th>Average</th>
<th>Half Width</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production On</td>
<td>0.9984</td>
<td>(Insufficient)</td>
<td>0</td>
<td>1.0000</td>
</tr>
<tr>
<td>Stock on Hand</td>
<td>103.95</td>
<td>17.31494</td>
<td>0</td>
<td>500.18</td>
</tr>
</tbody>
</table>

### Other

<table>
<thead>
<tr>
<th>Output</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lost Percentage</td>
<td>0.2518</td>
</tr>
</tbody>
</table>
### Frequencies Report

**5:37:20PM**

**Frequencies**

February 15, 2001

<table>
<thead>
<tr>
<th>Replication 1</th>
<th>Pack Process</th>
<th>Replications: 1</th>
</tr>
</thead>
</table>

### Start Time: 0.00  Stop Time: 1,000,000.00  Time Units: Minutes

<table>
<thead>
<tr>
<th>Process States</th>
<th>Number Obs</th>
<th>Average Time</th>
<th>Standard Percent</th>
<th>Restricted Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSY</td>
<td>3,366</td>
<td>204.90</td>
<td>68.97</td>
<td>68.97</td>
</tr>
<tr>
<td>FAILED</td>
<td>3,365</td>
<td>92.2189</td>
<td>31.03</td>
<td>31.03</td>
</tr>
</tbody>
</table>
Experimentation and Analysis

• Clearly, the customer service level (probability that the demand of an arriving customer is fully satisfied) is quite low as measured by the complementary probability of partially or fully unsatisfied demands, which was estimated by the simulation as 0.25 (the value of *Lost Percentage* in the *Output* section of the *User Specified* report)

• We want to modify the system to increase the customer service level (equivalently, we want to decrease the loss probability from 0.25 to an “acceptable” level)

  in inventory-oriented systems, such as the one under consideration, the only way to increase the customer service level is to increase the level of inventory on-hand
Improvement Strategy 1

• In our case, we may attempt to achieve the goal of increasing the customer service level is to use **Strategy 1** for modifying the original Production/Inventory system as follows:
  • invest more in maintenance activities
  • this would **reduce downtimes** and consequently make the process more available for production

• The next two reports show the improvements under this strategy (**Strategy 1**)
  • the average repair time is **reduced** from 90 minutes to 70 minutes
  • consequently, the loss probability is **reduced** from 0.25 to 0.08!
The *Frequencies* report below indicates that *Strategy 1* reduces the average repair time to 70 minutes with a standard deviation of 25 minutes.

<table>
<thead>
<tr>
<th>Process States</th>
<th>Number Obs</th>
<th>Average Time</th>
<th>Standard Percent</th>
<th>Restricted Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSY</td>
<td>3,603</td>
<td>206.59</td>
<td>74.43</td>
<td>74.43</td>
</tr>
<tr>
<td>FAILED</td>
<td>3,602</td>
<td>70.9755</td>
<td>25.57</td>
<td>25.57</td>
</tr>
</tbody>
</table>
**User Specified Report for Strategy 1**

- The *User Specified* report below indicates that **Strategy 1** reduces the loss probability to 0.08.

---

<table>
<thead>
<tr>
<th>Pack Process</th>
<th>Replications: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication 1</td>
<td>Start Time: 0.00  Stop Time: 1,000,000.00  Time Units: Minutes</td>
</tr>
</tbody>
</table>

**Tally**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Average</th>
<th>Half Width</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount Lost Per Customer</td>
<td>21.4974</td>
<td>(Insufficient)</td>
<td>0.3449</td>
<td>83.9904</td>
</tr>
</tbody>
</table>

**Time Persistent**

<table>
<thead>
<tr>
<th>Time Persistent</th>
<th>Average</th>
<th>Half Width</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production On</td>
<td>0.9805</td>
<td>(Insufficient)</td>
<td>0</td>
<td>1.0000</td>
</tr>
<tr>
<td>Stock on Hand</td>
<td>176.44</td>
<td>16.61622</td>
<td>0</td>
<td>504.84</td>
</tr>
</tbody>
</table>

**Other**

<table>
<thead>
<tr>
<th>Output</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lost Percentage</td>
<td>0.08431195</td>
</tr>
</tbody>
</table>
Improvement Strategy 2

• **Strategy 2** increases the reorder level so that the reorder level is hit sooner and production would resume earlier
  - when the reorder level is doubled, then the amount lost per customers is marginally improved from 25% to about 21.5%
  - however, we would like to do better as in Strategy 1

• This analysis leads us to conclude the following:

  a significant improvement in the customer service level can be achieved by improving the production process, rather than modifying the inventory replenishment policy
Time-Dependent Arrivals

• So far we 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 processes are stationary.

• However, it is quite common in practice for the underlying probability law to vary in time, in which case the underlying process is non-stationary (time dependent).
  • for example, rush hour traffic as opposed to ebb hour traffic.

• The Arena Create module provides a Schedule option in its Time Between Arrivals section, to be used only for single parameter distributions.
  • example: the exponential distribution, which gives rise to the Poisson arrival process.
Schedule Option of *Create* Module

Dialog box of a *Create* module (bottom) and spreadsheet view of its *Schedule* option (top)
Schedule Option Specification

Dialog box for a schedule specifying time-dependent arrival rates
**Schedule Option Specification (Cont.)**

Dialog box for the *Options*… field of a schedule