

SIMULATION MODELING AND ANALYSIS WITH ARENA

T. Altiok and B. Melamed

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

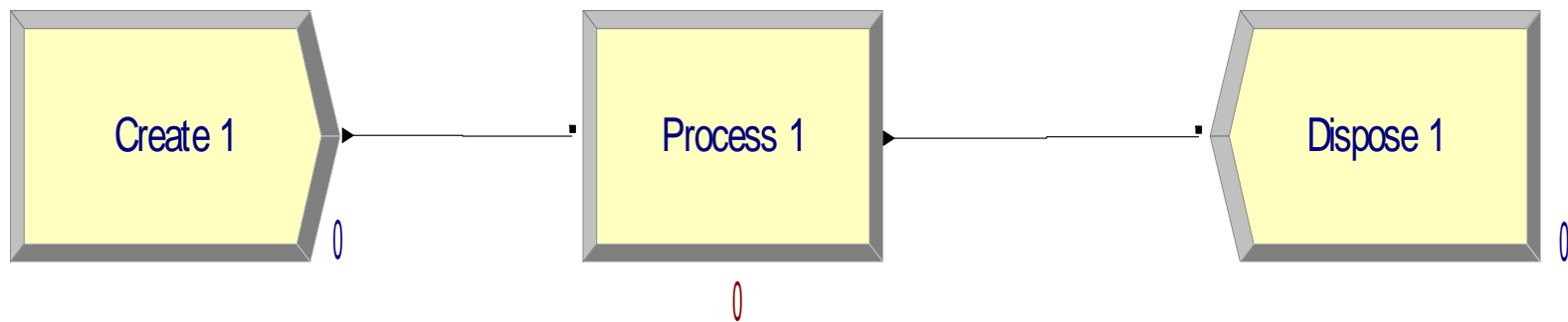
The screenshot shows the Arena Home Screen with the following components labeled:

- Title Bar:** Located at the top left of the window.
- Menu Bar:** Located below the title bar, containing File, Edit, View, Tools, Arrange, Object, Window, and Help.
- Standard Toolbar:** Located below the menu bar, containing various icons for file operations and editing.
- Project Bar Toolbar:** Located on the left side, containing icons for project management.
- Template Panel:** Located on the left side, containing a list of process templates such as Create, Process, Batch, Assign, Entity, Resource, Dispose, Decide, Separate, Record, Queue, and Variable.
- Model Window Canvas Flowchart View:** The central area showing a flowchart with three elements: Create 2, Process 1, and Dispose 1.
- Model Window Canvas Spreadsheet View:** Located at the bottom, showing a table with the following data:

	Name	Entity Type	Type	Value	Units	Entities per Arrival	Max Arrivals	First Creation
1	Create 2	Entity 1	Random	1	Hours	1	Infinite	0.0
- Run Interaction Toolbar:** Located at the top center, containing icons for simulation control.
- Drawing Toolbar:** Located below the Run Interaction Toolbar, containing icons for drawing and editing.
- Animate Toolbar:** Located on the right side, containing icons for animation.
- Animate Transfer Toolbar:** Located on the right side, containing icons for animation transfer.
- View Toolbar:** Located at the top right, containing icons for view manipulation.

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



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.)

The screenshot shows the 'Create' dialog box with the following settings:

- Name:** Create 1
- Entity Type:** Entity 1
- Time Between Arrivals:**
 - Type:** Expression
 - Expression:** EXPO(30)
 - Units:** Minutes
- Entities per Arrival:** 1
- Max Arrivals:** Infinite
- First Creation:** 0.0

Buttons: OK, Cancel, Help

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 screenshot shows the 'Process' dialog box with the following settings:

- Name:** Process 1
- Type:** Standard
- Logic:**
 - Action:** Seize Delay Release
 - Priority:** Medium(2)
 - Resources:** Resource, Machine, 1 (selected); <End of list>
 - Buttons:** Add... (disabled), Edit..., Delete
- Delay Type:** Expression
- Units:** Minutes
- Allocation:** Value Added
- Expression:** EXPO(24)

Buttons at the bottom: OK, Cancel, Help.

Dialog box for a *Process* module

Process Module (Cont.)

- 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

Arena Files

- 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)

Arena Simulation Results

- 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

5:28:06PM

Resources

March 2, 2001

A Work Station

Replications: 1

Replication 1

Start Time:

0.00

Stop Time:

600,000.00

Time Units: Minutes

MACHINE

Usage	Average	Half Width	Minimum	Maximum
Number Busy	0.7976	0.016547523	0	1.0000
Number Scheduled	1.0000	(Insufficient)	1.0000	1.0000
Utilization	0.7976	0.016547523	0	1.0000
Other		Value		
Number Times Used	19,890.00			
Scheduled Utilization	0.7976			

***Resources* statistics from a single replication
of the simple workstation model**

Example: a *Queues* Report

5:28:39PM

Queues

March 2, 2001

A Work Station

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	98.3230	14.85368	0	947.77
Other	Average	Half Width	Minimum	Maximum
Number Waiting	3.1937	0.532298979	0	42.0000

***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 *DAVG(S)* (the time average of the time-persistent statistic *S*), *TAVG(S)* (the average of tally element *S*), *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

The dialog box is titled "Record" and contains the following fields and options:

- Name:** Record 1
- Type:** Count (with a dropdown menu open showing: Count, Entity Statistics, Time Interval, Time Between, Expression)
- Value:** 1
- Counter Name:** Record 1

Buttons: OK, Cancel, Help

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

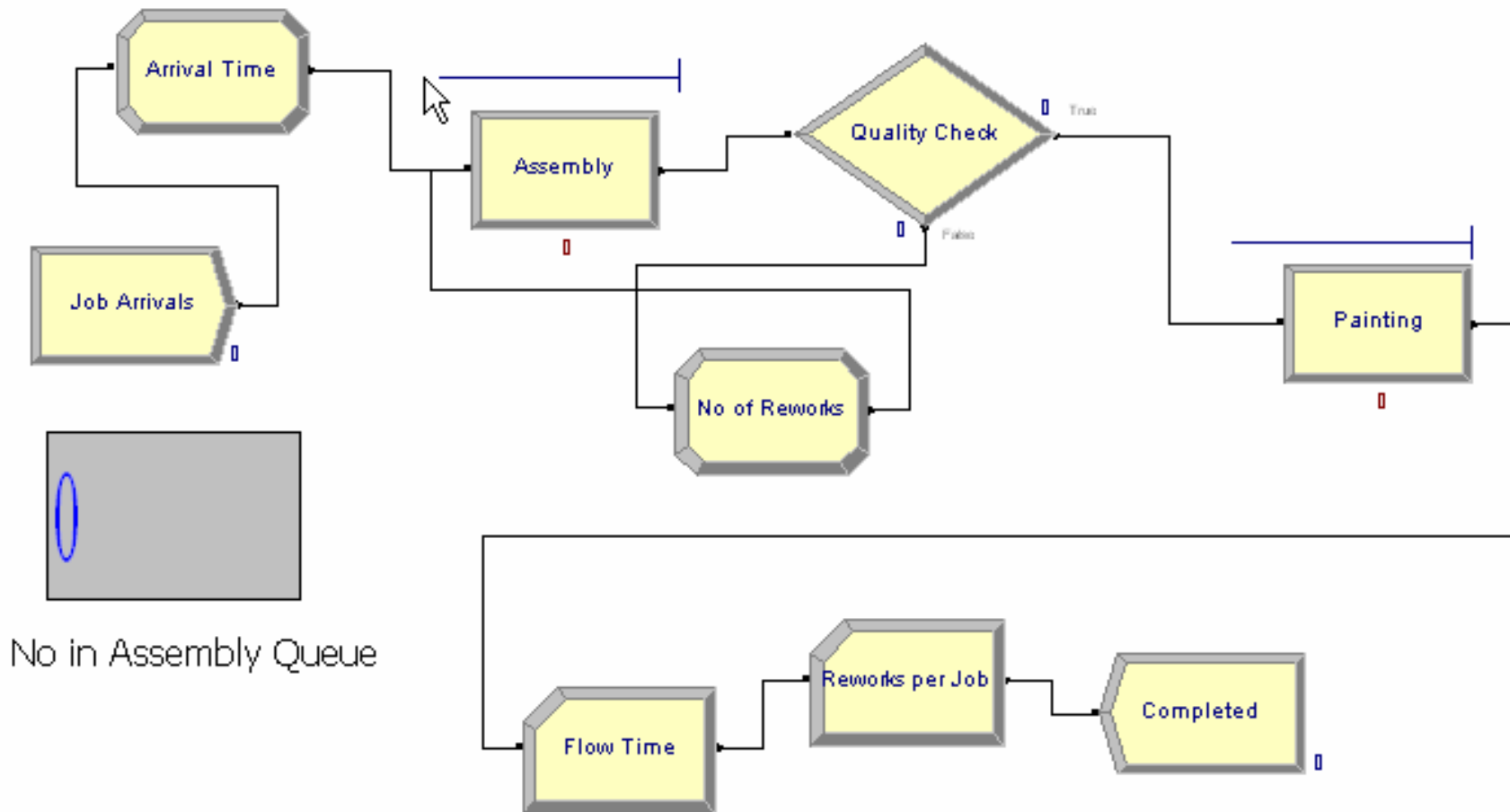
- Arena output reports consist of two types: **automatic** and **user-specified**
- An **automatic** report of summary statistics is generated automatically at the end of a simulation run by a number of Arena constructs, such as *entities*, *queues* and *resources*
 - those statistics are implicitly 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

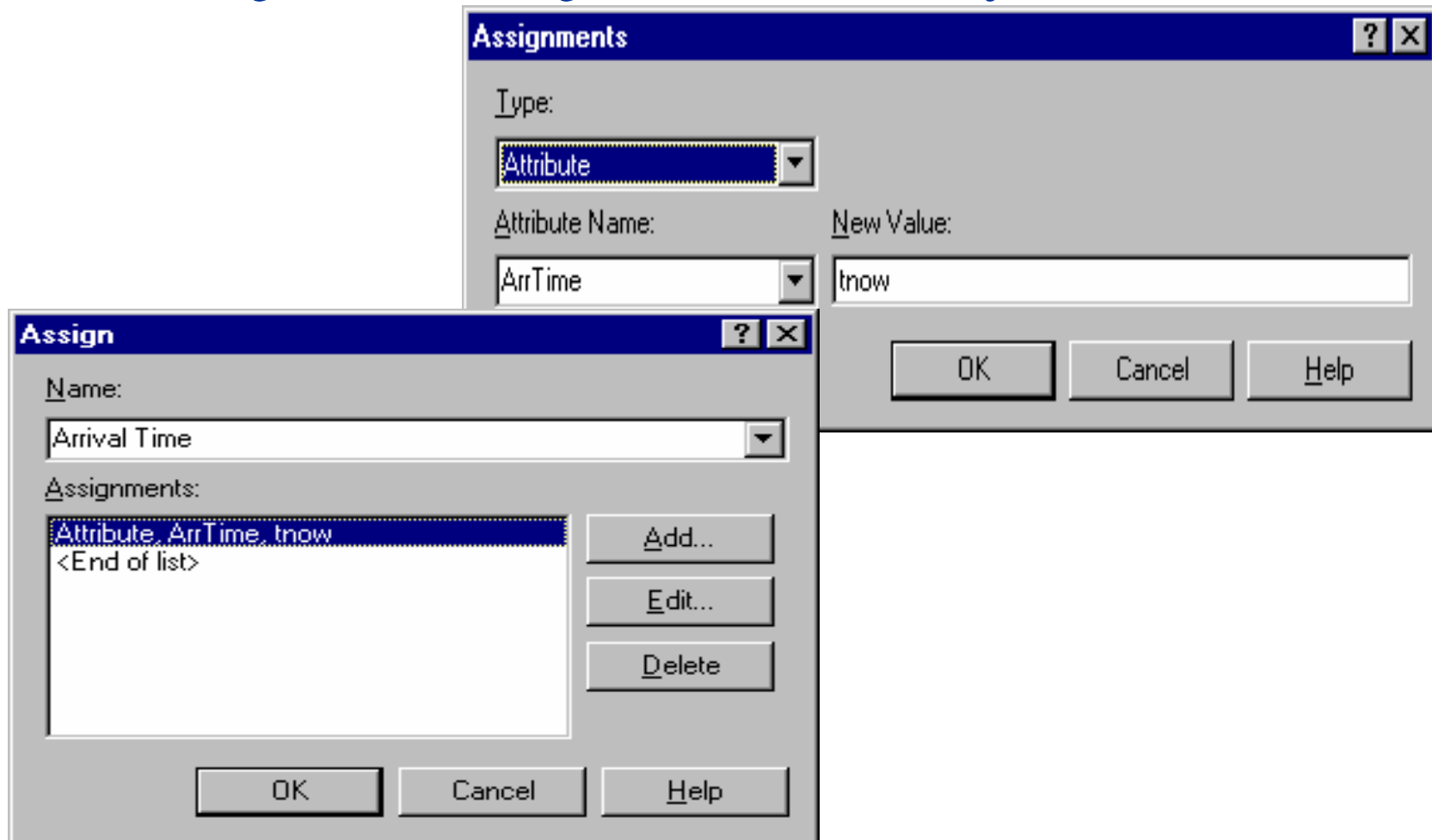
Two Processes in Series



Arena manufacturing model of assembly and painting processes in series

Assign Module

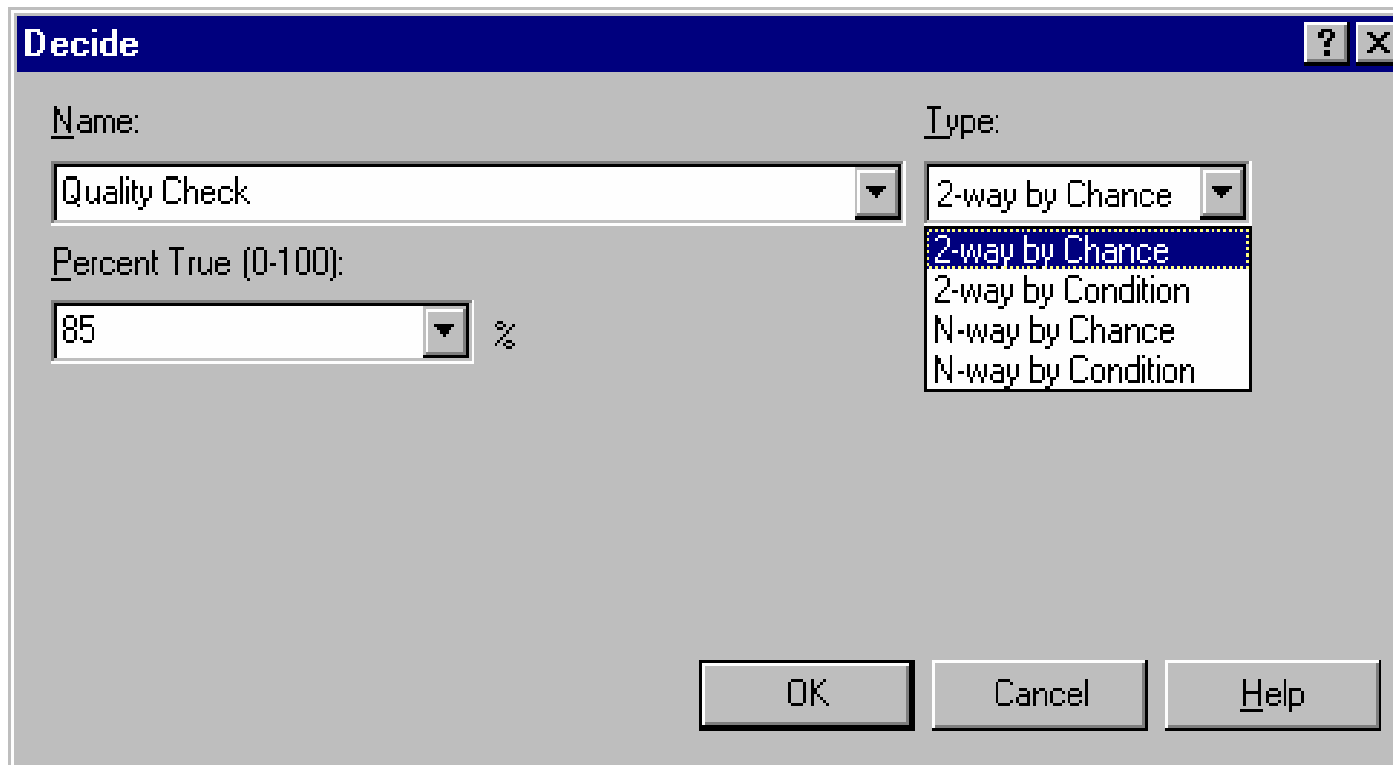
- The *Assign* module is used by *entities* to assign values to attributes
 - the dialog box below assigns an arrival time to a job attribute



Dialog boxes for an *Assign* module

Decide Module

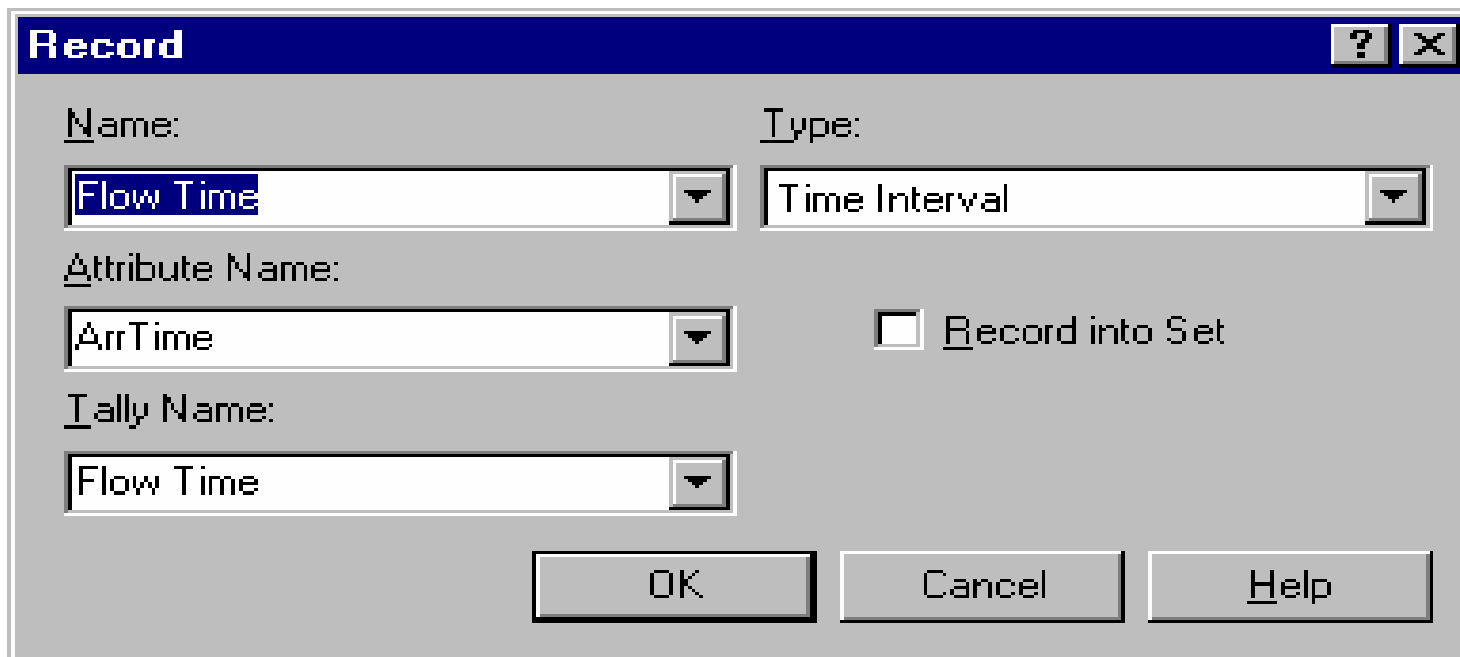
- The *Decide* module is used by *entities* to make branching decisions, based on chance or the truth/falsity of prescribed conditions
 - the dialog box below makes a two-way probabilistic branching decision



Dialog box for a *Decide* module

Record Module

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



The image shows a screenshot of the 'Record' dialog box in the Arena simulation software. The dialog box has a title bar with the word 'Record' and standard window controls (minimize, maximize, close). The main area contains several fields and a checkbox:

- Name:** A dropdown menu with 'Flow Time' selected.
- Type:** A dropdown menu with 'Time Interval' selected.
- Attribute Name:** A dropdown menu with 'ArrTime' selected.
- Tally Name:** A dropdown menu with 'Flow Time' selected.
- Record into Set:** An unchecked checkbox.

At the bottom of the dialog box are three buttons: 'OK', 'Cancel', and 'Help'.

Dialog box for a *Record* module

Resources Report

12:20:50PM

Resources

May 24, 2000

Assembly Op.

Replications: 1

Replication 1

Start Time:

0.00

Stop Time:

100,000.00

Time Units: Hours

Resource Detail Summary

Usage

	Number Busy	Number Scheduled	Utilization
ASSEMBLER	0.59	1.00	0.59
PAINTER	0.38	1.00	0.38

Other

	Number Times Used	Scheduled Utilization
ASSEMBLER	14741.00	0.59
PAINTER	12525.00	0.38

Queues Report

12:20:03PM

Queues

May 24, 2000

Assembly Op.

Replications: 1

Replication 1

Start Time:

0.00

Stop Time:

100,000.00

Time Units:

Hours

Queue Detail Summary

Time

	Waiting Time
Assembly.Queue	3.49
Painting.Queue	0.08
Total	3.57

Other

	Number Waiting
Assembly.Queue	0.51
Painting.Queue	0.01
Total	0.53

User Specified Report

12:21:28PM

User Specified

May 24, 2000

Assembly Op.

Replications: 1

Replication 1

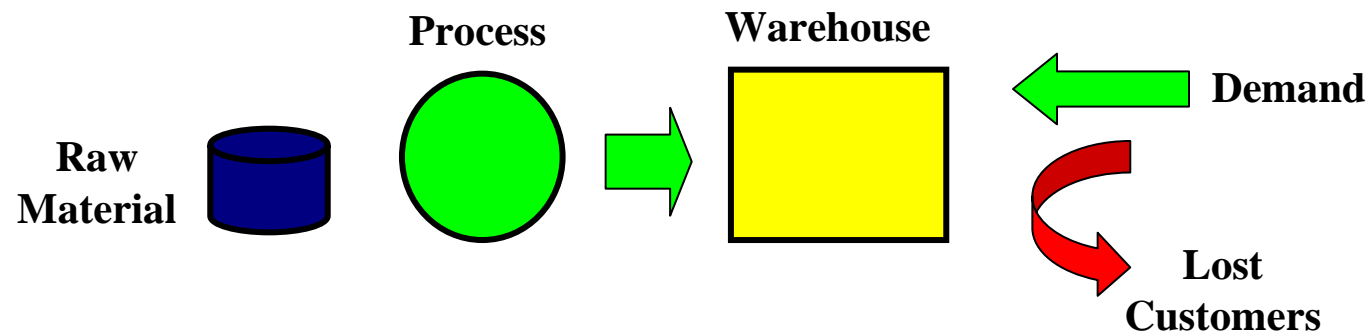
Start Time: 0.00 Stop Time: 100,000.00 Time Units: Hours

Tally

Interval	Average	Half Width	Minimum	Maximum
Flow Time	11.8873	0.495720704	5.0000	130.42

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

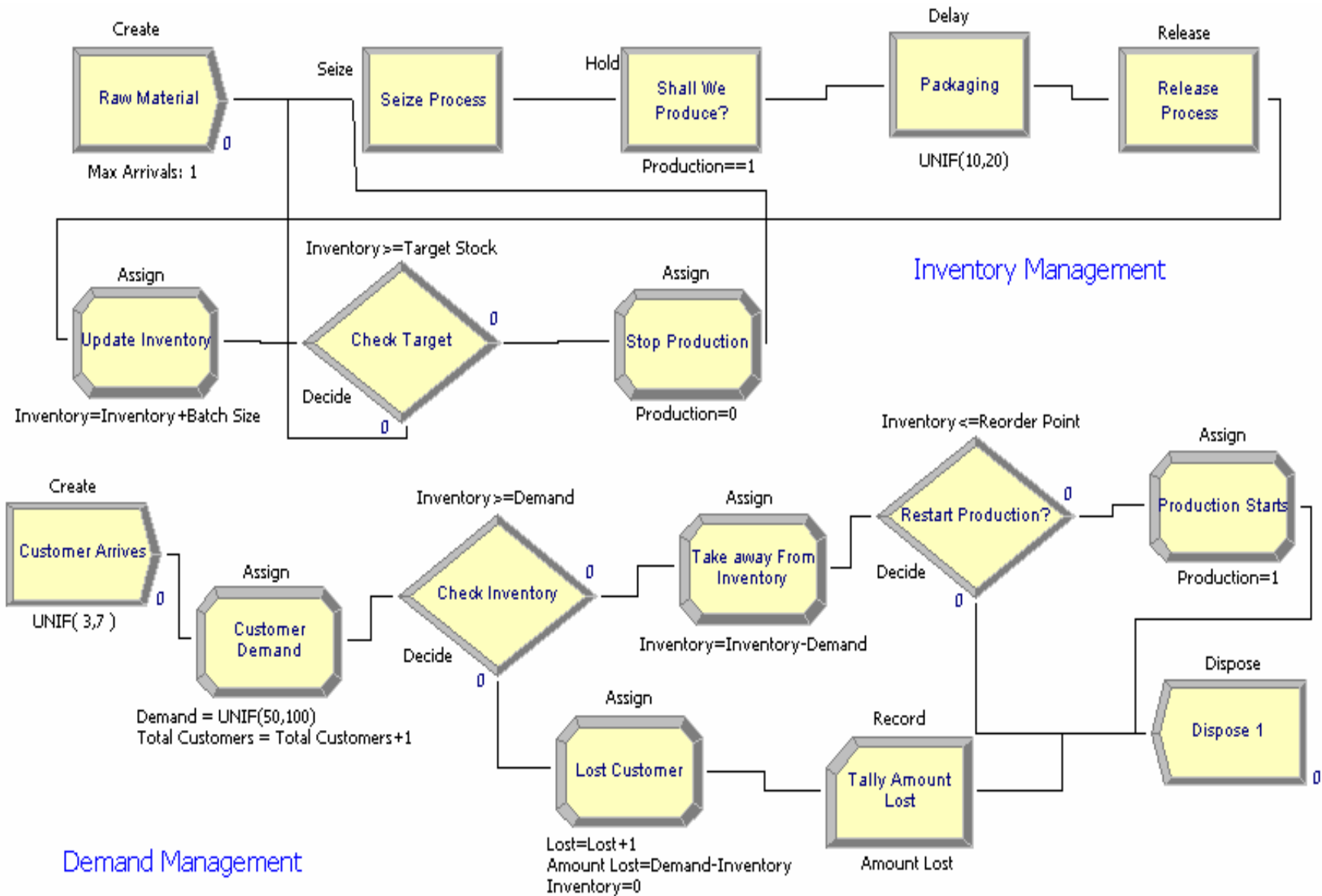
The Warehouse Operation

- 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



Arena Model Logic

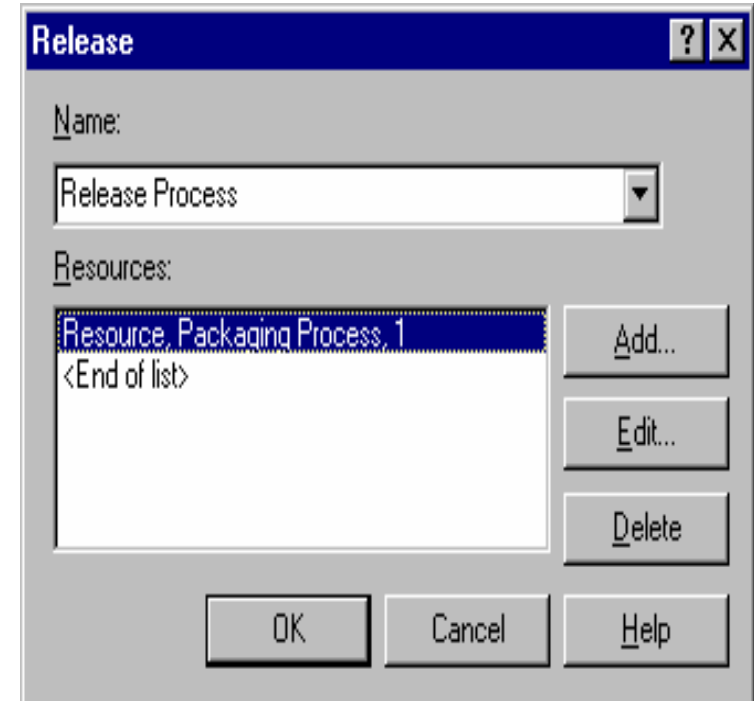
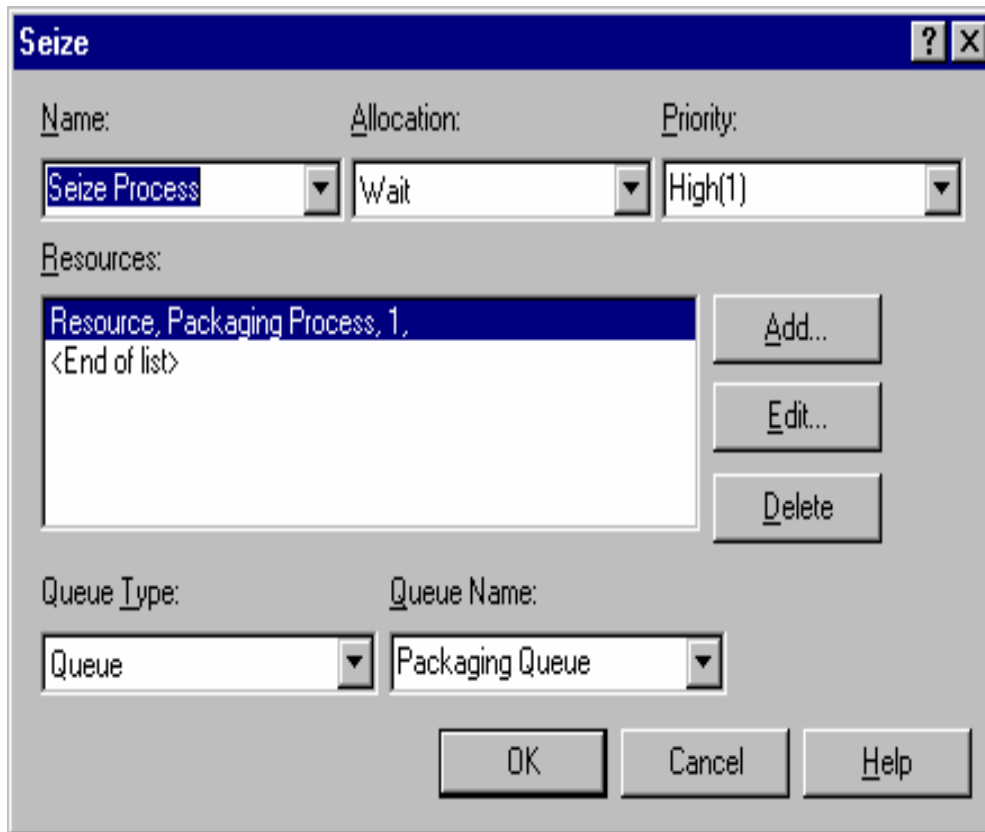
- The Arena model of the production/inventory system is composed of two segments: **inventory management** and **demand management**
- In the Arena model logic of the **inventory management segment**:
 - the packaging process takes a unit of raw material from its queue, and processes it as a batch of 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 **Unif**(10,20) minutes

Seize and Release Modules

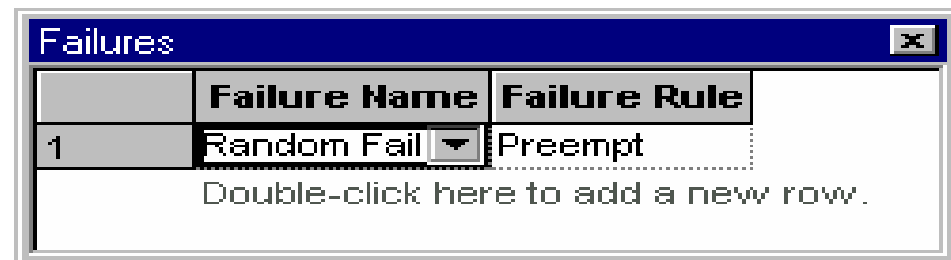
- The *Seize* module is used by *entities* to seize resources
- The *Release* module is used by *entities* to release resources
 - the dialog boxes below seize and release a packaging machine



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



Resource - Basic Process							
	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	Failures
1	Packaging Process	Fixed Capacity	1	0.0	0.0	0.0	1 rows

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

Failure Module

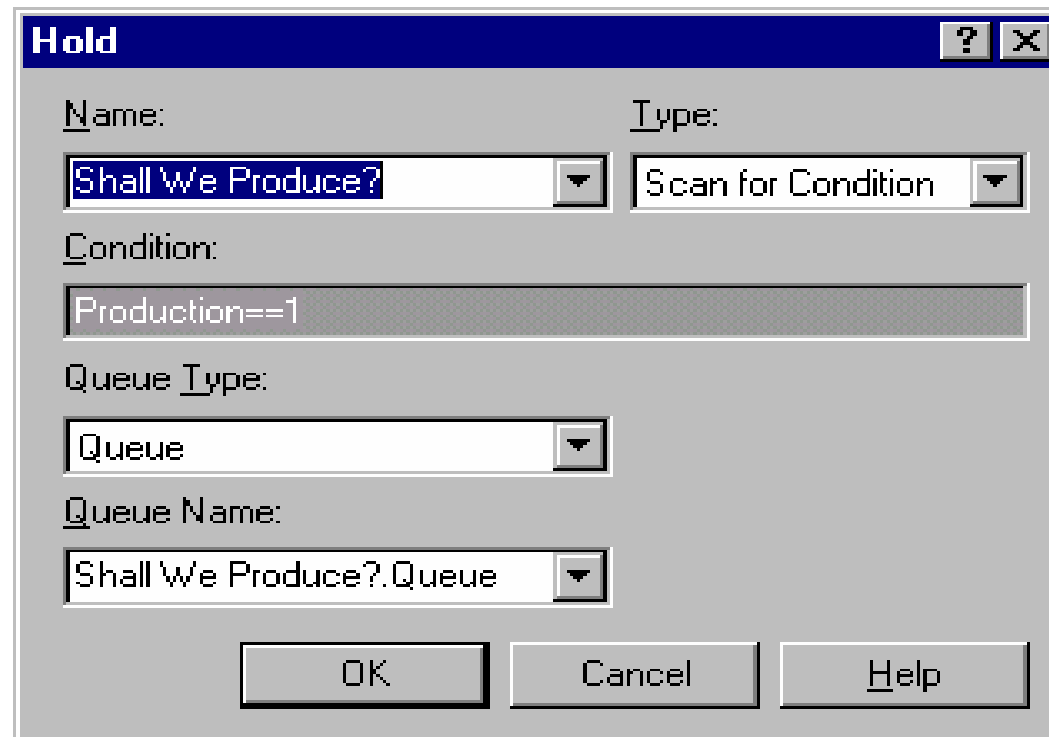
- All failure/repair data (uptimes and downtimes) are specified in the spreadsheet *Failure* module (from the *Advanced Process* template panel)
 - the dialog box below illustrates the data for a random failure

Failure - Advanced Process						
	Name	Type	Up Time	Up Time Units	Down Time	Down Time Units
1	Random Fail	Time	EXPO(200)	Minutes	NORM(70,45) ▼	Minutes

Dialog box of a *Failure* module

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

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

The screenshot shows the 'Decide' dialog box with the following configuration:

- Name:** Check Target
- Type:** 2-way by Condition
- If:** Variable
- Named:** Inventory
- Is:** >=
- Value:** Target Stock

Buttons: OK, Cancel, Help

Dialog Box of a *Decide* module

Decide Module (Cont.)

- The *entity* tests there 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

The screenshot shows the 'Create' dialog box with the following settings:

- Name:** Customer Arrives
- Entity Type:** Customer
- Time Between Arrivals:**
 - Type:** Expression
 - Expression:** UNIF(3.7)
 - Units:** Hours
- Entities per Arrival:** 1
- Max Arrivals:** Infinite
- First Creation:** 0.0

Buttons: OK, Cancel, Help

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 **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 **Unif(50,100)**

Decide Module

- 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 the 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 ($Lost = Lost + I$), 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

Variable - Basic Process						
	Name	Rows	Columns	Clear Option	Initial Values	Statistics
1	Inventory			System	1 rows	<input type="checkbox"/>
2	Batch Size			System	1 rows	<input type="checkbox"/>
3	Target Stock			System	1 rows	<input type="checkbox"/>
4	Production			System	1 rows	<input type="checkbox"/>
5	Reorder Point			System	1 rows	<input type="checkbox"/>
6	Demand			System	0 rows	<input type="checkbox"/>
7	Lost			System	0 rows	<input type="checkbox"/>
8	Total Customers			System	0 rows	<input type="checkbox"/>

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 bellow lists statistics collected in our example system

Statistic - Advanced Process								
	Name	Type	Expression	Report Label	Frequency Type	Resource Name	Report Label	Categories
1	Stock on Hand	Time-Persistent	Inventory	Stock on Hand	Value		Stock on Hand	0 rows
2	Process States	Frequency		Process States	State	Packaging Process	Process States	0 rows
3	Production On	Time-Persistent	Production==1	Production On	Value		Production On	0 rows
4	Lost Percentage	Output	Lost/Total Customers	Lost Percentage	Value		Lost Percentage	0 rows


Dialog Box of a *Statistic* module

Statistic Module (Cont.)

- 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 bellow specifies the tallying of demand loss at a *Record* module, called *Tally Lost Amount*



The image shows a screenshot of the 'Record' dialog box in Arena simulation software. The dialog box has a title bar with the text 'Record' and standard window controls (minimize, maximize, close). The main area contains several fields and controls:

- Name:** A dropdown menu with 'Tally Amount Lost' selected.
- Type:** A dropdown menu with 'Expression' selected.
- Value:** A text input field containing 'Amount Lost'.
- Record into Set:** An unchecked checkbox.
- Tally Name:** A dropdown menu with 'Avg Amount Lost Per Customer' selected.
- Buttons:** 'OK', 'Cancel', and 'Help' buttons are located at the bottom.

Dialog Box of a *Record* module

Record Module (Cont.)

- Whenever a customer *entity* enters the module, the expression (in this case, the variable) *Amount Lost* is evaluated, and the resultant value is tallied
- When the replication terminates, the output report will contain a *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

5:36:08PM

User Specified

February 15, 2001

Pack Process

Replications: 1

Replication 1

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

Tally

Expression	Average	Half Width	Minimum	Maximum
Amount Lost Per Customer	25.1504	1.48315	0.00985116	85.5513

Time Persistent

Time Persistent	Average	Half Width	Minimum	Maximum
Production On	0.9984	(Insufficient)	0	1.0000
Stock on Hand	103.95	17.31494	0	500.18

Other

Output	Value
Lost Percentage	0.2518

Frequencies Report

5:37:20PM

Frequencies

February 15, 2001

Pack Process

Replications: 1

Replication 1

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

Process States	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	3,366	204.90	68.97	68.97
FAILED	3,365	92.2189	31.03	31.03

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!

Frequencies Report for Strategy 1

- The *Frequencies* report below indicates that **Strategy 1** reduces the average repair time to 70 minutes with a standard deviation of 25 minutes

6:59:36PM

Frequencies

February 15, 2001

Pack Process

Replications: 1

Replication 1

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

Process States	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	3,603	206.59	74.43	74.43
FAILED	3,602	70.9755	25.57	25.57

User Specified Report for Strategy 1

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

6:58:11PM

User Specified

February 15, 2001

Pack Process

Replications: 1

Replication 1

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

Tally

Expression	Average	Half Width	Minimum	Maximum
Amount Lost Per Customer	21.4974	(Insufficient)	0.3449	83.9904

Time Persistent

Time Persistent	Average	Half Width	Minimum	Maximum
Production On	0.9805	(Insufficient)	0	1.0000
Stock on Hand	176.44	16.61622	0	504.84

Other

Output	Value
Lost Percentage	0.08431195

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

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

Create [?] [X]

Name: Create 1 Entity Type: Entity 1

Time Between Arrivals:

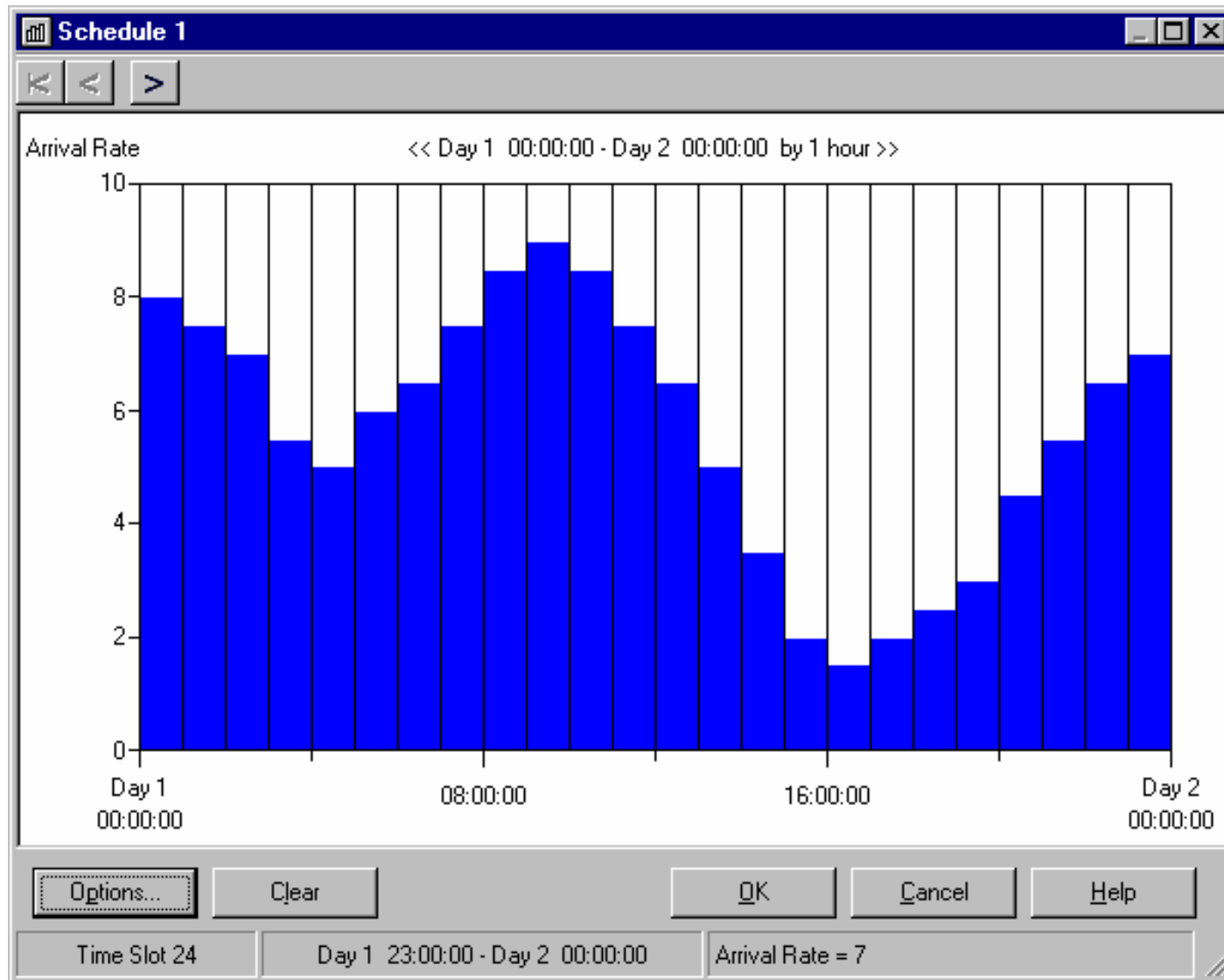
Type: Schedule Schedule Name: Schedule 1

Entities per Arrival: 1 Max Arrivals: Infinite

OK Cancel Help

**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.)

The screenshot shows a dialog box titled "Options" with a close button (X) in the top right corner. The dialog is divided into two main sections: "X-axis" and "Y-axis".

X-axis section:

- Time slot duration:** A dropdown menu showing "1 hour".
- Range (time slots):** A text input field containing "24".
- Calendar speed (slots per click):** A text input field containing "1".
- Show vertical grid lines**

Y-axis section:

- Maximum:** A text input field containing "10".
- Minimum:** A text input field containing "0".
- Snap spacing:** A text input field containing "0.5".

When at end of schedule:

- Repeat from beginning**
- Remain at arrival rate** (with a text input field containing "0" next to it)

At the bottom of the dialog are two buttons: "Apply" and "Cancel".

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