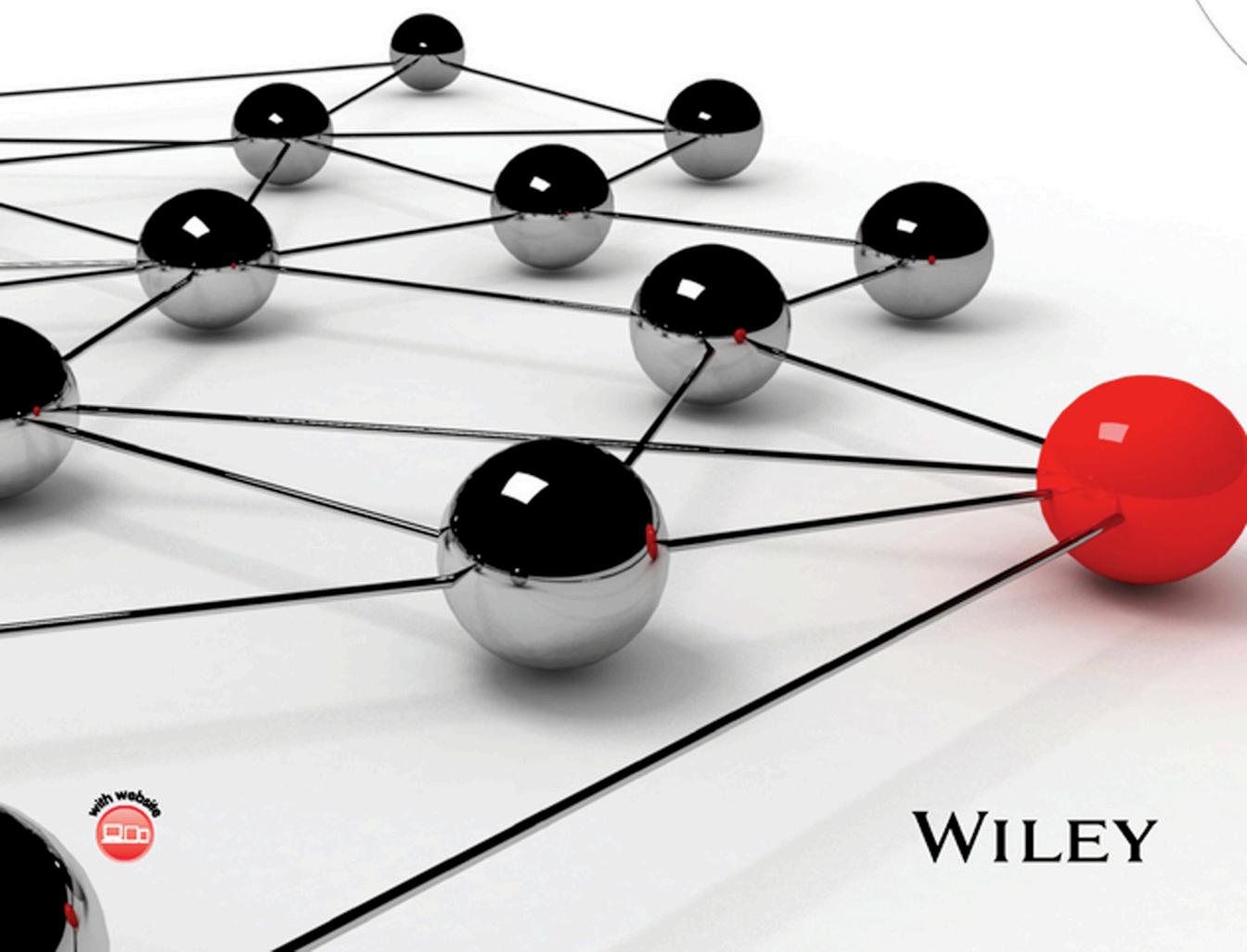


SECOND EDITION

# SIMULATION MODELING AND ARENA®

● MANUEL D. ROSSETTI ●



WILEY



## **SIMULATION MODELING AND ARENA®**



# **SIMULATION MODELING AND ARENA®**

---

**MANUEL D. ROSSETTI**

University of Arkansas

**WILEY**

Copyright © 2016 by John Wiley & Sons, Inc. All rights reserved

Published by John Wiley & Sons, Inc., Hoboken, New Jersey  
Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at [www.copyright.com](http://www.copyright.com). Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permission>.

**Limit of Liability/Disclaimer of Warranty:** While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at [www.wiley.com](http://www.wiley.com).

***Library of Congress Cataloging-in-Publication Data:***

Rossetti, Manuel D. (Manuel David), 1962- author.  
Simulation modeling and Arena / Manuel D. Rossetti. – Second edition.  
pages cm  
Includes bibliographical references and index.  
ISBN 978-1-118-60791-6 (cloth)  
1. Computer simulation. 2. Arena (Computer file) I. Title.  
QA76.9.C65R66 2015  
003'.3–dc23

2015006381

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

2 2016

*To all my students!*



# BRIEF TABLE OF CONTENTS

<b>1</b>	<b>Simulation Modeling</b>	<b>1</b>
<b>2</b>	<b>Generating Randomness in Simulation</b>	<b>17</b>
<b>3</b>	<b>Spreadsheet Simulation</b>	<b>63</b>
<b>4</b>	<b>Introduction to Simulation in Arena®</b>	<b>97</b>
<b>5</b>	<b>Basic Process Modeling</b>	<b>163</b>
<b>6</b>	<b>Modeling Randomness in Simulation</b>	<b>233</b>
<b>7</b>	<b>Analyzing Simulation Output</b>	<b>299</b>
<b>8</b>	<b>Modeling Queuing and Inventory Systems</b>	<b>393</b>
<b>9</b>	<b>Entity Movement and Material-Handling Constructs</b>	<b>489</b>
<b>10</b>	<b>Miscellaneous Topics in Arena Modeling</b>	<b>543</b>
<b>11</b>	<b>Application of Simulation Modeling</b>	<b>633</b>



# CONTENTS

<b>Preface</b>	<b>xv</b>
<b>Acknowledgments</b>	<b>xvii</b>
<b>Introduction</b>	<b>xix</b>
<b>1 Simulation Modeling</b>	<b>1</b>
1.1 Simulation Modeling, 1	
1.2 Why Simulate? 2	
1.3 Types of Computer Simulation, 3	
1.4 Simulation: Descriptive or Prescriptive Modeling? 6	
1.5 Randomness in Simulation, 7	
1.6 Simulation Languages, 7	
1.7 Simulation Methodology, 8	
1.8 Organization of The Book, 15	
Exercises, 15	
<b>2 Generating Randomness in Simulation</b>	<b>17</b>
2.1 The Stochastic Nature of Simulation, 17	
2.2 Random Numbers, 18	
2.3 Random Number Generators, 19	
2.4 Testing Random Numbers, 24	
2.4.1 Distributional Tests, 25	
2.4.2 Testing Independence, 34	
2.5 Generating Random Variates from Distributions, 37	
2.5.1 Inverse Transform, 38	
2.5.2 Convolution, 46	
2.5.3 Acceptance/Rejection, 47	

2.5.4	Mixture Distributions, Truncated Distributions, and Shifted Random Variables,	50	
2.6	Summary,	54	
	Exercises,	54	
<b>3</b>	<b>Spreadsheet Simulation</b>		<b>63</b>
3.1	Simulation in a Spreadsheet Environment,	63	
3.2	Useful Spreadsheet Functions and Methods,	64	
3.2.1	Using RAND() and RANDBETWEEN(),	64	
3.2.2	Using VLOOKUP(),	66	
3.2.3	Using Data Tables to Repeatedly Sample,	68	
3.2.4	Using VBA,	69	
3.3	Example Spreadsheet Simulations,	70	
3.3.1	Simple Monte-Carlo Integration,	71	
3.3.2	The Classic News Vendor Inventory Problem,	73	
3.3.3	Simulating a Random Cash Flow,	76	
3.4	Introductory Statistical Concepts,	79	
3.4.1	Point Estimates and Confidence Intervals,	79	
3.4.2	Determining the Sample Size,	81	
3.5	Summary,	86	
	Exercises,	86	
<b>4</b>	<b>Introduction to Simulation in Arena®</b>		<b>97</b>
4.1	Introduction,	97	
4.2	The Arena Environment,	98	
4.3	Performing Simple Monte-Carlo Simulations Using Arena,	100	
4.3.1	Redoing Area Estimation with Arena,	101	
4.3.2	Redoing the News Vendor Problem with Arena,	104	
4.4	How The Discrete-Event Clock Works,	106	
4.5	Modeling a Simple Discrete-Event Dynamic System,	110	
4.5.1	A Drive-through Pharmacy,	110	
4.5.2	Modeling the System,	110	
4.5.3	Implementing the Model in Arena,	113	
4.5.4	Specify the Arrival Process,	114	
4.5.5	Specify the Resources,	116	
4.5.6	Specify the Process,	116	
4.5.7	Specify Run Parameters,	118	
4.5.8	Analyze the Results,	120	
4.6	Extending the drive through pharmacy model,	123	
4.7	Animating the Drive-Through Pharmacy Model,	125	
4.8	Getting Help in Arena,	133	
4.9	Siman and The Run Controller,	133	
4.9.1	SIMAN MOD and EXP Files,	134	
4.9.2	Using the Run Controller,	138	
4.10	How Arena Manages Entities and Events,	145	
4.11	Summary,	149	
	Exercises,	150	

<b>5 Basic Process Modeling</b>	<b>163</b>
5.1 Elements of Process-Oriented Simulation,	163
5.2 Entities, Attributes, and Variables,	164
5.3 Creating and Disposing of Entities,	165
5.4 Defining Variables and Attributes,	169
5.5 Processing Entities,	174
5.6 Attributes, Variables, and Some I/O,	176
5.6.1 Modifying the Pharmacy Model,	176
5.6.2 Using the ASSIGN Module,	180
5.6.3 Using the READWRITE Module,	181
5.6.4 Using the RECORD Module,	184
5.6.5 Animating a Variable,	186
5.6.6 Running the Model,	187
5.7 Flow of Control in Arena,	190
5.7.1 Logical and Probabilistic Conditions,	191
5.7.2 Iterative Looping,	195
5.7.3 Example: Iterative Looping, Expressions, and Submodels,	196
5.8 Batching and Separating Entities,	210
5.8.1 Example: Tie-Dye T-Shirts,	210
5.9 Summary,	221
Exercises,	223
<b>6 Modeling Randomness in Simulation</b>	<b>233</b>
6.1 Random Variables and Probability Distributions,	233
6.2 Modeling with Discrete Distributions,	238
6.3 Modeling with Continuous Distributions,	240
6.4 Input Distribution Modeling,	242
6.5 Fitting Discrete Distributions,	244
6.5.1 Fitting a Poisson Distribution,	244
6.5.2 Visualizing the Data,	245
6.5.3 Statistical Analysis of the Data,	247
6.5.4 Checking the Goodness of Fit of the Model,	250
6.6 Fitting Continuous Distributions,	254
6.6.1 Visualizing the Data,	255
6.6.2 Statistically Summarize the Data,	256
6.6.3 Hypothesizing and Testing a Distribution,	257
6.6.4 Visualizing the Fit,	263
6.7 Using The Input Analyzer,	267
6.8 Additional Input Modeling Concepts,	276
6.9 Modeling Randomness in Arena,	279
6.9.1 Conceptualizing the Model,	280
6.9.2 Implementing the Model,	282
6.10 Summary,	292
Exercises,	293

<b>7 Analyzing Simulation Output</b>	<b>299</b>
7.1 Types of Statistical Variables, 300	
7.2 Types of Simulation with Respect to Output Analysis, 305	
7.3 Analysis of Finite-Horizon Simulations, 307	
7.3.1 Determining the Number of Replications, 309	
7.3.2 Finite Horizon Example, 311	
7.3.3 Sequential Sampling for Finite-Horizon Simulations, 318	
7.4 Analysis of Infinite-Horizon Simulations, 321	
7.4.1 Assessing the Effect of Initial Conditions, 327	
7.4.2 Performing the Method of Replication–Deletion, 332	
7.4.3 Looking for the Warm-Up Period in the Output Analyzer, 335	
7.4.4 The Method of Batch Means, 346	
7.4.5 Performing the Method of Batch Means, 350	
7.5 Comparing System Configurations, 353	
7.5.1 Comparing Two Systems, 354	
7.5.2 Analyzing Multiple Systems, 372	
7.6 Summary, 382	
Exercises, 384	
<b>8 Modeling Queuing and Inventory Systems</b>	<b>393</b>
8.1 Introduction, 393	
8.2 Single Line Queuing Stations, 394	
8.2.1 Queuing Notation, 396	
8.2.2 Little’s Formula, 398	
8.2.3 Deriving Formulas for Markovian Single-Queue Systems, 401	
8.3 Examples and Applications of Queuing Analysis, 407	
8.3.1 Infinite Queue Examples, 407	
8.3.2 Finite Queue Examples, 412	
8.4 Non-Markovian Queues and Approximations, 417	
8.5 Simulating Single Queues in Arena, 419	
8.5.1 Machine Interference Optimization Model, 419	
8.5.2 Using OptQuest on the Machine Interference Model, 424	
8.5.3 Modeling Balking and Reneging, 427	
8.6 Holding and Signaling Entities, 435	
8.6.1 Redoing the M/M/1 Model with HOLD/SIGNAL, 437	
8.7 Networks of Queuing Stations, 442	
8.7.1 STATION, ROUTE, and SEQUENCE Modules, 444	
8.8 Inventory Systems, 453	
8.8.1 Modeling an $(r, Q)$ Inventory Control Policy, 454	
8.8.2 Modeling a Multi-Echelon Inventory System, 464	
8.9 Summary, 471	
Exercises, 472	

<b>9 Entity Movement and Material-Handling Constructs</b>	<b>489</b>
9.1 Introduction, 489	
9.2 Resource-Constrained Transfer, 490	
9.2.1 Implementing Resource-Constrained Transfer, 492	
9.2.2 Animating Resource-Constrained Transfer, 498	
9.3 Constrained Transfer with Transporters, 501	
9.3.1 Test and Repair Shop with Workers as Transporters, 504	
9.3.2 Animating Transporters, 509	
9.4 Modeling Systems with Conveyors, 511	
9.4.1 Test and Repair Shop with Conveyors, 516	
9.4.2 Animating Conveyors, 519	
9.4.3 Miscellaneous Issues in Conveyor Modeling, 522	
9.5 Modeling Guided Path Transporters, 528	
9.6 Summary, 537	
Exercises, 537	
<b>10 Miscellaneous Topics in Arena Modeling</b>	<b>543</b>
10.1 Introduction, 543	
10.2 Non-stationary Processes, 544	
10.2.1 Thinning Method, 547	
10.2.2 Rate Inversion Method, 548	
10.3 Advanced Resource Modeling, 552	
10.3.1 Scheduled Capacity Changes, 553	
10.3.2 Calculating Utilization, 559	
10.3.3 Resource Failure Modeling, 562	
10.4 Tabulating Frequencies Using the Statistic Module, 565	
10.5 Resource and Entity Costing, 568	
10.5.1 Resource Costing, 568	
10.5.2 Entity Costing, 571	
10.6 Miscellaneous Modeling Concepts, 576	
10.6.1 Picking Between Stations, 576	
10.6.2 Generic Station Modeling, 579	
10.6.3 Picking up and Dropping Off Entities, 585	
10.7 Programming Concepts Within Arena, 593	
10.7.1 Using the Generated Access File, 593	
10.7.2 Working with Files, Excel, and Access, 596	
10.7.3 Using Visual Basic for Applications, 609	
10.7.4 Generating Correlated Random Variates, 622	
10.8 Summary, 625	
Exercises, 625	

<b>11 Application of Simulation Modeling</b>	<b>633</b>
11.1 Introduction, 633	
11.2 SM Testing Contest Problem Description, 635	
11.3 Answering the Basic Modeling Questions, 640	
11.4 Detailed Modeling, 645	
11.4.1 Conveyor and Station Modeling, 646	
11.4.2 Modeling Samples and the Test Cells, 648	
11.4.3 Modeling Sample Holders and the Load/Unload Area, 655	
11.4.4 Performance Measure Modeling, 658	
11.4.5 Simulation Horizon and Run Parameters, 659	
11.4.6 Preliminary Experimental Analysis, 663	
11.5 Final Experimental Analysis and Results, 663	
11.5.1 Using the Process Analyzer on the Problem, 664	
11.5.2 Using OptQuest on the Problem, 668	
11.5.3 Investigating the New Logic Alternative, 671	
11.6 Sensitivity Analysis, 671	
11.7 Completing the Project, 672	
11.8 Some Final Thoughts, 675	
Exercises, 676	
<b>Bibliography</b>	<b>677</b>
<b>Appendix A Common Distributions</b>	<b>683</b>
<b>Appendix B Statistical Tables</b>	<b>689</b>
<b>Appendix C Distributions, Operators, Functions in Arena</b>	<b>697</b>
<b>Appendix D Queuing Theory Formulas</b>	<b>699</b>
<b>Appendix E Inventory Theory Formulas</b>	<b>703</b>
<b>Appendix F Useful Equations</b>	<b>705</b>
<b>Appendix G Arena Panel Modules</b>	<b>707</b>
<b>Index</b>	<b>711</b>

# PREFACE

Welcome to the second edition. Similar to the first edition, the book is intended as an introductory textbook for a first course in discrete-event simulation modeling and analysis for upper-level undergraduate students as well as entering graduate students. While the text is focused toward engineering students (primarily industrial engineering), it could also be utilized by advanced business majors, computer science majors, and other disciplines where simulation is practiced. Practitioners interested in learning simulation and Arena® could also use this book independently of a course.

The second edition has been significantly reorganized to allow more introductory material involving the application of spreadsheets to perform simulation. By including spreadsheet simulation, students are able to experience introductory concepts such as random number generation and sampling in a more familiar venue. Then, the concepts can be reinforced using Arena and discrete-event modeling. The book also introduces the use of the open-source statistical package, R, both for performing statistical testing and for fitting distributions. A significant number of additional exercises have been added to each chapter.

MANUEL D. ROSSETTI  
Fayetteville, AR



## **ACKNOWLEDGMENTS**

I would like to thank my children, Joseph and Maria, who gave me their support and understanding, and my wife, Amy, who not only gave me support, but also helped in creating figures and diagrams and proofreading. Thank you so much!

M.D.R



# INTRODUCTION

## INTENDED AUDIENCE

Discrete-event simulation is an important tool for the modeling of complex system. It is used to represent manufacturing, transportation, and service systems in a computer program for the purpose of performing experiments. The representation of the system via a computer program enables the testing of engineering design changes without disruption to the system being modeled. Simulation modeling involves elements of system modeling, computer programming, probability and statistics, and engineering design. Because simulation modeling involves these individually challenging topics, the teaching and learning of simulation modeling can be difficult for both instructors and students. Instructors are faced with the task of presenting computer programming concepts, probability modeling, and statistical analysis all within the context of teaching how to model complex systems such as factories and supply chains. In addition, because of the complexity associated with simulation modeling, specialized computer languages are needed and thus must be taught to students for use during the model building process. This book is intended to help instructors with this daunting task.

Traditionally, there have been two primary types of simulation textbooks: (i) those that emphasize the theoretical (and mostly statistical) aspects of simulation and (ii) those that emphasize the simulation language or package. The intention of this book is to blend these two aspects of simulation textbooks together while adding and emphasizing the art of model building. Thus the book contains chapters on modeling and chapters that emphasize the statistical aspects of simulation. However, the coverage of statistical analysis is integrated with the modeling in such a way to emphasize the importance of both the topics. This book utilizes the Arena® simulation environment as the primary modeling tool for teaching simulation. Arena is one of the leading simulation modeling packages in the world and has a strong and active user base. While the book uses Arena as the primary modeling tool, the book is not intended to be a user's guide to Arena. Instead, Arena is used as the vehicle for explaining important simulation concepts.

I feel strongly that simulation is best learned by doing. The book is structured to enable and encourage students to get engaged in the material. The overall approach to presenting the material is based on a hands-on concept for student learning. The style of writing is informal, tutorial, and centered around examples that students can implement while reading the chapters. The book assumes a basic knowledge of probability and statistics and an introductory knowledge of computer programming. Even though these topics are assumed, the book provides integrated material that should refresh students on the basics of these topics. Thus, instructors who use this book should not have to formally cover this material and can be assured that students who read the book will be aware of these concepts within the context of simulation.

## ORGANIZATION OF THE BOOK

Chapter 1 is an introduction to the field of simulation modeling and to modeling methodologies. After Chapter 1, the student should know what simulation is and be able to put the different types of simulation into context. It also describes an overall simulation methodology that has been proven useful both in the classroom and in practice.

Chapter 2 presents the theory and practice of generating random numbers and random variates from probability distributions. In addition, the student will know why random number generators and their control are essential for simulation modeling. The testing of pseudorandom numbers is also presented. After this chapter, the student should be aware of the methods used to generate and test pseudorandom number sequences. In addition, several methods of generating random variates are discussed (inverse transform, acceptance/rejection, convolution, and composition). These techniques are introduced without the added burden of computer methods.

The focus of Chapter 3 is on introducing Monte-Carlo methods within the context of a spreadsheet environment. Students will learn to put into practice the random number and random variate generation methods discussed in Chapter 2. In addition, the chapter presents materials on how to organize and perform basic simulation methods within a spreadsheet. The notions of sampling, sample size determination, and statistical inference are introduced.

Chapter 4 introduces Arena. First, Arena is used to perform simple Monte-Carlo simulation, similar to what was presented in Chapter 3. Chapter 4 also introduces the important concept of how a discrete-event clock ticks and sets the stage for process modeling using activity diagramming. A simple (but comprehensive) example of Arena is presented so that students will feel comfortable with the tool. Finally, debugging and how Arena processes events and entities are discussed.

Chapter 5 dives deeper into process-oriented modeling. The statistical aspects of simulation are downplayed within the chapter. The Basic Process template within Arena is thoroughly covered. Important concepts within process-oriented modeling (e.g., entities, attributes, activities, and state variables) are discussed within the context of a number of examples. In addition, a deeper understanding of Arena is developed including flow of control, input/output, variables, arrays, and debugging. After finishing Chapter 5, the student should be able to model interesting systems from a process viewpoint using Arena.

Chapter 6 emphasizes the role of randomness in simulation. After Chapter 6, the student should be able to model the input distributions required for simulation using tools such as R and the Arena Input Analyzer.

Building on the use of stochastic elements in simulation, Chapter 7 discusses the major methods by which simulation output analysis must account for randomness. The different types of statistical quantities (observation based versus time persistent) are defined and then statistical methods are introduced for their analysis. Specifically, the chapter covers the method of replication for finite-horizon simulations, the analysis of the initialization transient period, the method of replication–deletion, and the method of batch means. In addition, the use of simulation to make decisions between competing alternatives is presented.

Chapter 8 returns to model building by presenting models for important classic modeling situations in queuing and inventory theory. Both analytical and simulation approaches to modeling these systems are covered. For those instructors who work in a curriculum that has a separate course on these topics, this chapter presents an opportunity to concentrate on simulating these systems. The analytical material could easily be skipped without loss of continuity; however, often students learn the most about these systems through simulation. For those instructors where this material is not covered separately, background is presented on these topics to ensure that students can apply the basics of queuing theory and are aware of basic inventory models. Then, the basic models are extended so that students understand how simulation quickly becomes necessary when modeling more realistic situations.

Chapter 9 presents a thorough treatment of the entity transfer and material handling constructs within Arena. Students learn the fundamentals of resource-constrained transfers, free path transporters, conveyors, and fixed path transporters. The animation of models containing these elements is also emphasized.

Chapter 10 pulls together a number of miscellaneous topics that round out the use of Arena. In particular, the chapter covers activity-based costing and presents advanced aspects of modeling with resources (e.g., schedules and failure modeling). It also presents a few useful modules that were not previously covered (e.g., picking stations, generic stations, and picking up and dropping off entities). An introduction to using Visual Basic and Arena is also presented.

Finally, Chapter 11 presents a detailed case study using Arena. An IIE/Rockwell Software Arena Contest problem is solved in its entirety. This chapter ensures that students will be ready to solve such a problem if assigned as a project for the course. The chapter wraps up with some practical advice for performing simulation projects.

## SPECIAL FEATURES

- Each chapter begins with specific learning objectives.
- Integrating the statistical aspects of simulation (e.g., output analysis) with the tool. More detailed discussions of the statistical aspects of simulation are presented than is found in many other simulation language-oriented textbooks.
- Studies have shown that activity-based learning is critical to student retention of material. The text is organized around the building of models with the intention

that students should be following along at the computer while working through the chapters. Instructors can perform the activities or organize computer laboratory exercises around the development of the models in the text.

- Special emphasis on the computer programming aspects of simulation: Students who take a course based on this text will be expected to have had at least one entry-level computer programming course; however, even with this background, most students are woefully ill-prepared to use computers to solve problems. The theory-based textbooks do not cover this material and the simulation package textbooks attempt to downplay the programming aspects of their environment so that the modeling environment appears attractive to noncomputer-oriented users. This book is intended to enable students to understand the inner workings of the simulation environment and thus demystify the black box. The language elements of the simulation environment are compared to standard computer language elements so that students can make the appropriate analogies to already studied material. Detailed presentation of pseudo-code enhances the understanding of the programming constructs.
- While Arena is the modeling tool, the conceptual modeling process presented in the text is based on language-independent methods including but not limited to rich picturing, elementary flowcharting, activity diagramming, and pseudo-code development. The emphasis is placed on developing a specification for a model that could be implemented in any simulation language environment.
- Coverage of classic stochastic models from operations research: One chapter is dedicated to queuing and inventory models. In many curricula, if the analytical models are presented, they will be taught in a different course. In my opinion, the simulation of classic models along with their analytical treatment can provide for deeper student learning on these topics. In addition, the presentation of these classic models both analytically and through simulation provides simple systems on which to build the teaching of complex more practical extensions.
- Comprehensive examples, exercises, questions, and problem sets developed from the authors' teaching, research, and industrial experiences.
- Students can download the Arena software directly from [www.arenasimulation.com](http://www.arenasimulation.com). Chapter illustrations and files (e.g., Arena and Microsoft® Office Excel®) are available from the course site. A comprehensive set of presentation slides are also available for students and instructors to use within the classroom. Finally, solutions for many of the exercises are available to instructors.

## COURSE SYLLABUS SUGGESTION

Early versions of the manuscript for this textbook were used for multiple semesters in my course at the University of Arkansas. The course that I teach is to junior/senior-level undergraduate industrial engineering students. In addition, graduate students who have never had a course in simulation take the course. Graduate students are given extra homework assignments and are tested over some of the more theoretical aspects presented in the text (e.g., acceptance/rejection). I am able to cover most parts of Chapters Chapter 1–9 within a typical 16-week semester offering. A typical topic outline is as follows assuming 80-minute lecture periods.

#Lectures	Topic	Readings
1	Introduction	Chapter 1
2	Random number generation	Chapter 2
2	Random variate generation	Chapter 2
2	Spreadsheet Simulation	Chapter 3
2	Introduction to Arena	Chapter 4
4	Basic Process Modeling	Chapter 5
2	Input modeling	Chapter 6
2	Finite-horizon simulation	Chapter 7
3	Infinite-horizon simulation	Chapter 7
1	Comparing alternatives	Chapter 7
4	Queuing models	Chapter 8
5	Entity transfer and material handling constructs	Chapter 9
30		

I use three examinations and a project within the course. Examination 1 covers Chapters Chapter 1–3. Examination 2 covers Chapters Chapter 4–6. Examination 3 tests over Chapters Chapter 7–9. I do not formally test the students on the material in Chapters Chapter 10 and Chapter 11 since they will be using all previously learned material and components when performing their final project. Students are assigned homework throughout the semester and quizzed on the homework via online questions. In addition, to formal lectures, my course has a computer-based laboratory component that meets 1 day each week. During this time, the students are required to work on computer-based assignments that are based on the examples in the textbook.

## ABOUT THE AUTHOR

Dr. Manuel D. Rossetti grew up in Canton, Ohio. He received his PhD and MSIE degrees in Industrial and Systems Engineering from The Ohio State University and his BSIE degree from the University of Cincinnati. He is a registered professional engineer in the State of Arkansas.

Dr. Rossetti joined the Industrial Engineering Faculty at the University of Arkansas in August 1999. He was awarded tenure and promoted to Associated Professor in August 2003 and promoted to Full Professor in August 2010. Dr. Rossetti was also appointed Associate Department Head for the Department of Industrial Engineering in August 2010. Dr. Rossetti currently serves as the Director for the Center for Excellence in Logistics and Distribution, a National Science Foundation Industry/University Cooperative Research Center.

His research interests include the design, analysis, and optimization of manufacturing, health care, logistics, and transportation systems using stochastic modeling, computer simulation, and mathematical modeling techniques. Dr. Rossetti has published over 85 journal and conference articles in the areas of transportation, manufacturing, health care, and simulation, and he has obtained over \$4 million dollars in research funding. His research sponsors have included the Pine Bluff Arsenal, The Arkansas Science and Technology Authority, the Defense Logistics Agency, the Naval Systems Supply Command, the Air Force Research Laboratory, Wal-Mart Stores Inc., the Transportation Research Board, US Department of Transportation, and the National Science Foundation.

Dr. Rossetti was selected as a Lilly Teaching Fellow funded by the Lilly Endowment in 1997/1998 and won a UA Baum Teaching Enhancement Grant in 2002. Dr. Rossetti has won the UA Industrial Engineering Outstanding Teaching Award in 2002, 2008, and 2011. He has also been voted as the best IE teacher by IE students in 2007 and 2009. Dr. Rossetti received the John L. Imhoff Outstanding Teacher Award in 2011–2012 for the College of Engineering and the Charles and Nadine Baum Teaching Award for the University of Arkansas in 2013, the highest teaching honor bestowed at the university. In 2013, Dr. Rossetti was elected into the University of Arkansas' Teaching Academy.

---

# 1

---

## SIMULATION MODELING

### LEARNING OBJECTIVES

- To be able to describe what computer simulation is.
- To be able to discuss why simulation is an important analysis tool.
- To be able to list and describe the various types of computer simulations.
- To be able to describe a simulation methodology.

### 1.1 SIMULATION MODELING

In this book, you will learn how to model systems within a computer environment in order to analyze system design configurations. The models that you will build and exercise are called simulation models. When developing a simulation model, the modeler attempts to represent the system in such a way that the representation assumes or mimics the pertinent outward qualities of the system. This representation is called a simulation model. When you execute the simulation model, you are performing a simulation. In other words, simulation is an instantiation of the act of simulating. A simulation is often the next best thing to observing the real system. If you have confidence in your simulation, you can use it to infer how the real system will operate. You can then use your inference to understand and improve the system's performance.

In general, simulations can take on many forms. Almost everyone is familiar with the board game Life™. In this game, the players imitate life by going to college, getting a job, getting married, etc. and finally retiring. This board game is a simulation of life. As another

example, the military performs war game exercises which are simulations of battlefield conditions. Both of these simulations involve a physical representation of the thing being simulated. The board game, the rules, and the players represent the simulation model. The battlefield, the rules of engagement, and the combatants are also physical representations. No wishful thinking will make the simulations that you develop in this book real. This is the first rule to remember about simulation. A simulation is only a model (representation) of the real thing. You can make your simulations as realistic as time and technology allows, but they are not the real thing. As you would never confuse a toy airplane with a real airplane, you should never confuse a simulation of a system with the real system. You may laugh at this analogy, but as you apply simulation to the real world, you will see analysts who forget this rule. Don't be one.

All the previous examples involved a physical representation or model (real things simulating other real things). In this book, you will develop computer models that simulate real systems. Ravindran et al. [1987] defined computer simulation as “A numerical technique for conducting experiments on a digital computer which involves logical and mathematical relationships that interact to describe the behavior of a system over time.” Computer simulations provide an extra layer of abstraction from reality that allows fuller control of the progression of and the interaction with the simulation. In addition, even though computer simulations are one step removed from reality, they are often capable of providing constructs which cannot be incorporated into physical simulations. For example, an airplane flight simulator can have emergency conditions for which it would be too dangerous or costly to provide in a physical-based simulation training scenario. This representational power of computer modeling is one of the main reasons why computer simulation is used.

## 1.2 WHY SIMULATE?

Imagine trying to analyze the following situation. Patients arrive at an emergency room. The arrival of the patients to the emergency department occurs randomly and may vary with the day of the week and even the hour of the day. The hospital has a triage station, where the arriving patient's condition is monitored. If the patient's condition warrants immediate attention, the patient is expedited to an emergency room bed to be attended by a doctor and a nurse. In this case, the patient's admitting information may be obtained from a relative. If the patient does not require immediate attention, the patient goes through the admitting process, where the patient's information is obtained. The patient is then directed to the waiting room, to wait for allocation to a room, a doctor, and a nurse. The doctors and nurses within the emergency department must monitor the health of the patients by performing tests and diagnosing the patient's symptoms. This occurs on a periodic basis. As the patient receives care, the patient may be moved to and require other facilities [magnetic resonance imaging (MRI), X-ray, etc.]. Eventually, the patient is either discharged after receiving care or admitted to the main hospital. The hospital is interested in conducting a study of the emergency department in order to improve the care of the patients while better utilizing the available resources. To investigate this situation, you might need to understand the behavior of certain measures of performance:

- The average number of patients who are waiting.
- The average waiting time of the patients and their average total time in the emergency department.

- The average number of rooms required per hour.
- The average utilization of the doctors and nurses (and other equipment).

Because of the importance of emergency department operations, the hospital has historical records available on the operation of the department through its patient tracking system. With these records, you might be able to estimate the current performance of the emergency department. Despite the availability of this information, when conducting a study of the emergency department, you might want to propose changes to how the department will operate (e.g., staffing levels) in the future. Thus, you are faced with trying to predict the future behavior of the system and its performance when making changes to the system. In this situation, you cannot realistically experiment with the actual system without possibly endangering the lives or care of the patients. Thus, it would be better to model the system and test the effect of changes on the model. If the model has acceptable fidelity, then you can infer how the changes will affect the real system. This is where simulation techniques can be utilized.

If you are familiar with operations research and industrial engineering techniques, you may be thinking that the emergency department can be analyzed by using queuing models. Later chapters of this book will present more about queuing models; however, for the present situation, the application of queuing models will most likely be inadequate due to the complex policies for allocating nurses, doctors, and beds to the patients. In addition, the dynamic nature of this system (the non-stationary arrivals, changing staffing levels, etc.) cannot be well modeled with current analytical queuing models. Queuing models might be used to analyze portions of the system, but a total analysis of the dynamic behavior of the entire system is beyond the capability of these types of models. But, a total analysis of the system is not beyond simulation modeling.

A key advantage of simulation modeling is that it has the capability of modeling the entire system and its complex interrelationships. The representational power of simulation provides the flexible modeling that is required for capturing complex processes. As a result, all the important interactions among the different components of the system can be accounted for within the model. The modeling of these interactions is inherent in simulation modeling because simulation imitates the behavior of the real system (as closely as necessary). The prediction of the future behavior of the system is then achieved by monitoring the behavior of different modeling scenarios as a function of simulated time. Real-world systems are often too complex for analytical models and often too expensive to experiment with directly. Simulation models allow the modeling of this complexity and enable low cost experimentation to make inferences about how the actual system might behave.

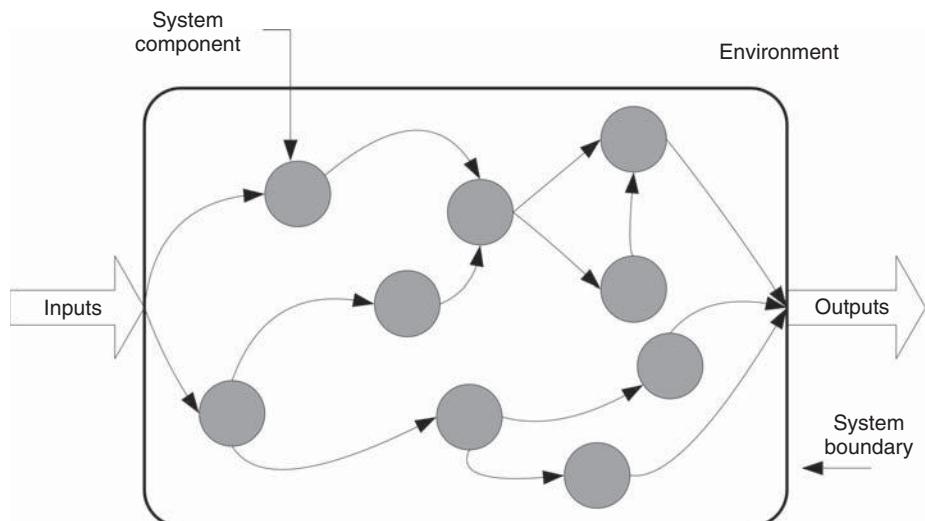
### 1.3 TYPES OF COMPUTER SIMULATION

The main purpose of a simulation model is to allow observations about a particular system to be collected as a function of time. So far the word *system* has been used in much of the discussion, without formally discussing what a system is. According to Blanchard and Fabrycky [1990], a system is a set of interrelated components working together toward a common objective. The standard for systems engineering provides a deeper definition:

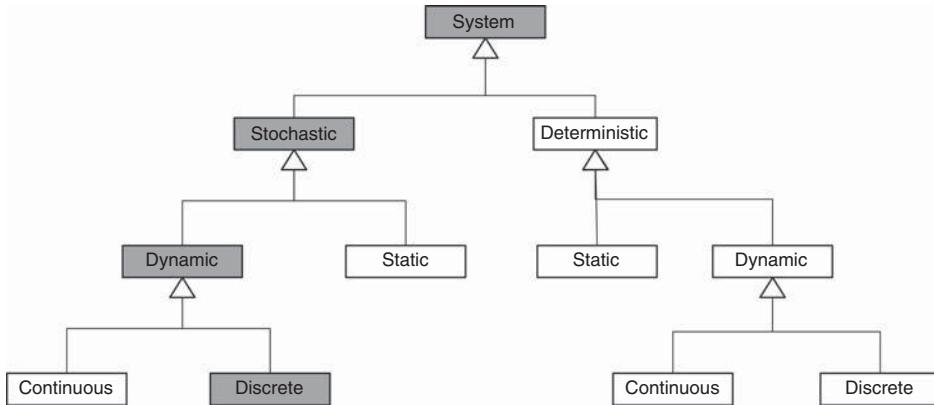
"A system is a composite of people, products, and processes that provide a capability to satisfy stated needs. A complete system includes the facilities, equipment (hardware and software), materials, services, data, skilled personnel, and techniques required to achieve, provide, and sustain system effectiveness." Air Force Systems Command (1991)

Figure 1.1 illustrates the fact that a system is embedded within an environment and that typically a system requires inputs and produces output using internal components. How you model a particular system will depend on the intended use of the model and how you perceive the system. The modeler's view of the system colors how they conceptualize it. For example, for the emergency room situation, "What are the system boundaries? Should the ambulance dispatching and delivery process be modeled? Should the details of the operating room be modeled?" Clearly, the emergency room has these components, but your conceptualization of it as a system may or may not include these items, and thus, your decisions regarding how to conceptualize the system will drive the level of abstraction within your modeling. An important point to remember is that two perfectly logical and rational people can look at the same thing and conceptualize that thing as two entirely different systems based on their "Weltanschauung" or world view.

Because how you conceptualize a system drives your modeling, it is useful to discuss some general system classifications. Systems might be classified by whether or not they are man-made (e.g., manufacturing system) or whether they are natural (e.g., solar system). A system can be physical (e.g., an airport) or conceptual (e.g., a system of equations). If stochastic or random behavior is an important component of the system, then the system is said to be stochastic; if not, then it is considered deterministic. One of the more useful ways to look at a system is whether it changes with respect to time. If a system does not change significantly with respect to time, it is said to be static, else it is called dynamic. If a system is dynamic, you might want to consider how it evolves with respect to time. A dynamic system is said to be discrete if the state of the system changes at discrete points in time. A dynamic system is said to be continuous if the state of the system changes continuously with time. This dichotomy is purely a function of your level of abstraction. If conceptualizing



**Figure 1.1** A conceptualization of a system.



**Figure 1.2** General types of systems.

a system as discrete, serves our purposes, then you can call the system discrete. Figure 1.2 illustrates this classification of systems. This book primarily examines stochastic, dynamic, discrete systems.

The main purpose of a simulation model is to allow observations about a particular system to be gathered as a function of time. From that standpoint, there are two distinct types of simulation models: (i) discrete event and (ii) continuous.

Just as discrete systems change at discrete points in time, in a discrete-event simulation, observations are gathered at selected points in time when certain changes take place in the system. These selected points in time are called events. On the other hand, continuous simulation requires that observations be collected continuously at every point in time (or at least that the system is described for all points in time). The types of models to be examined in this book are called discrete-event simulation models.

To illustrate the difference between the two types of simulation models, contrast a fast food service counter with that of oil loading facility that is filling tankers. In the fast food service counter system, changes in the status of the system occur either when a customer arrives to place an order or when the customer receives their food. At these two events, measures such as queue length and waiting time will be affected. At all the other points in time, these measures remain either unchanged (e.g., queue length) or not yet ready for observation (e.g., waiting time of the customer). For this reason, the system does not need to be observed on a continuous basis. The system need only be observed at selected discrete points in time, resulting in the applicability of a discrete-event simulation model.

In the case of the oil tanker loading example, one of the measures of performance is the amount of oil in each tanker. Because the oil is a liquid, it cannot be readily divided into discrete components. That is, it flows continuously into the tanker. It is not necessary (or practical) to track each molecule of oil individually when you only care about the level of the oil in the tanker. In this case, a model of the system must describe the rate of flow over time and the output of the model is presented as a function of time. Systems such as these are often modeled using differential equations. The solution of these equations involves numerical methods that integrate the state of the modeled system over time. This, in essence, involves dividing time into small equal intervals and stepping through time.

Often both the discrete and continuous viewpoints are relevant in modeling a system. For example, if oil tanker arrives at the port to be filled, we have an arrival event that changes

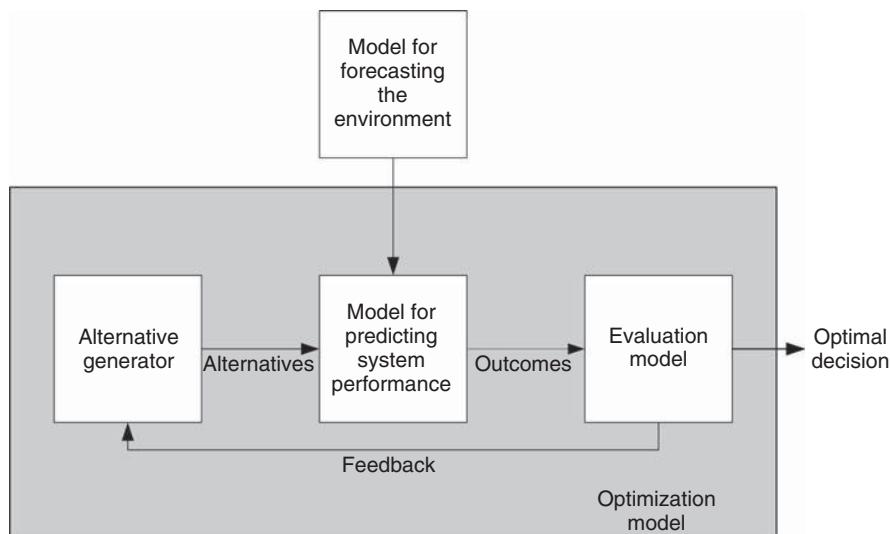
the state of the system. This type of modeling situation is called combined continuous discrete modeling. Some simulation languages have modeling constructs for both continuous and discrete modeling; however, this book does not cover the modeling of continuous or combined continuous discrete systems. There are many useful references on this topic. The focus of this book will be on discrete-event modeling.

#### 1.4 SIMULATION: DESCRIPTIVE OR PRESCRIPTIVE MODELING?

A descriptive model describes how a system behaves. Simulation is at its heart a descriptive modeling technique. Simulation is used to depict the behaviors or characteristics of existing or proposed systems. However, a key use of simulation is to convey the *required* behaviors or properties of a proposed system. In this situation, simulation is used to prescribe a solution. A prescriptive model tells us what to do. In other words, simulation can also be used for prescriptive modeling. Figure 1.3 illustrates the concept of using simulation to recommend a solution.

In the figure, a simulation model is used for predicting the behavior of the system. Input models are used to characterize the system and its environment. An evaluative model is used to evaluate the output of the simulation model to understand how the output compares to desired goals. The alternative generator is used to generate different scenarios to be fed into the simulation model for evaluation. Through a feedback mechanism the inputs can be changed based on the evaluation of the outputs and eventually a recommended solution can be achieved.

For example, in modeling a drive up pharmacy, suppose that the probability of a customer waiting longer than 3 minutes in line had to be less than 10%. To form design alternatives, the inputs (e.g., number of pharmacists, possibly the service process) can be varied. Each alternative can then be evaluated to see if the waiting time criteria are met. In this simple situation, you might act as your own alternative generator and the evaluative model is as



**Figure 1.3** Using simulation for prescriptive analysis.

simple as meeting a criteria; however, in more complex models, there will often be hundreds of inputs to vary and multiple competing objectives. In such situations, simulation optimization and heuristic search methods are often used. This is an active and important area of research within simulation.

## 1.5 RANDOMNESS IN SIMULATION

In most real-life situations, the arrival process and the service process occur in a random manner. Even though the processes may be random, it does not mean that you cannot describe or model the randomness. To have any hope of simulating the situation, you must be able to model the randomness. One of the ways to model this randomness is to describe the phenomenon as a random variable governed by a particular probability distribution. For example, if the arrivals to the bank occur according to a Poisson process, then from probability theory, it is known that the distribution of interarrival times is an exponential distribution. In general, information about how the customers arrive must be secured either through direct observation of the system or by using historical data. If neither source of information is available, then some plausible assumptions must be made to describe the random process by a probability model.

If historical data is available, there are two basic choices for how to handle the modeling. The first choice is to develop a probability model, given the data. The second choice is to try to drive the simulation directly from the historical data. The latter approach is not recommended. First of all, it is extremely unlikely that the captured data will be in a directly usable form. Secondly, it is even more unlikely that the data will be able to adequately represent all the modeling scenarios that you will need through the course of experimenting with the model. For example, suppose that you only have 1 day's worth of arrival data, but you need to simulate a month's worth of system operation. If you simply redrive your simulation using the 1 day's worth of data, you are not simulating different days! It is much more advisable to develop probability models either from historical data or from data that you capture in developing your model. Chapter 6 will discuss some of the tools and techniques for modeling probability distributions.

Once a probability model has been developed, statistical theory provides the means for obtaining random samples based on the use of uniformly distributed random numbers on the interval (0,1). These random samples are then used to map the future occurrence of an event on the time scale. For example, if the interarrival time is exponential, then a random sample drawn from that distribution would represent the time interval until the occurrence of the next arrival. The process of generating random numbers and random variables within simulation will be discussed in Chapter 2.

## 1.6 SIMULATION LANGUAGES

Discrete-event simulation normally involves a tremendous volume of computation. Consequently, the use of computers to carry out these computations is essential; however, the volume of computations is not the only obstacle in simulation. If you consider the bank teller example discussed in the previous sections, you will discover that it involves a complex logical structure that requires special expertise before it can be translated into a computer model. Attempting to implement the simulation model, from scratch, in a general-purpose language

such as FORTRAN, Visual Basic, C/C++, or Java will require above-average programming skills. In the absence of specialized libraries for these languages that try to relieve the user from some of the burden, simulation as a tool would be relegated to “elite” programmers. Luckily, the repetitive nature of computations in simulation allows the development of computer libraries that are applicable to simulation modeling situations. For example, libraries or packages must be available for ordering and processing events chronologically, as well as generating random numbers and automatically collecting statistics. Such a library for simulating discrete-event systems in Java is available from the author, see Rossetti [2008].

The computational power and storage capacity of computers have motivated the development of specialized simulation languages. Some languages have been developed for continuous or discrete simulations. Others can be used for combined continuous and discrete modeling. All simulation languages provide certain standard programming facilities and will differ in how the user will take advantage of these facilities. There is normally some trade-off between how flexible the language is in representing certain modeling situations. Usually, languages that are highly flexible in representing complex situations require more work (and care) by the user to account for how the model logic is developed. Some languages are more programming oriented (e.g., SIMSCRIPT<sup>TM</sup>) and others are more “drag and drop” (e.g., ProModel<sup>TM</sup>, Arena<sup>TM</sup>).

The choice of a simulation language is a difficult one. There are many competing languages, each having their own advantages and disadvantages. The Institute for Operations Research and Management Science (INFORMS) often has a yearly product review covering commercial simulation languages, see, for example, <http://lionhrtpub.com/orms/>. In addition to this useful comparison, you should examine the Winter Simulation Conference ([www.wintersim.org](http://www.wintersim.org)). The conference has hands on exhibits of simulation software and the conference proceedings often have tutorials for the various software packages. Past proceedings have been made available electronically through the generous support of the INFORMS Society for Simulation (<http://www.informs-sim.org/wscpapers.html>).

Arena<sup>TM</sup> was chosen for this textbook because of the author’s experience utilizing the software, its ease of use, and the availability of student versions of the software. While all languages have flaws, using a simulation language is essential in performing high performance simulation studies. Most, if not all simulation companies have strong support to assist the user in learning their software. Arena<sup>TM</sup> has a strong academic and industrial user base and is very competitive in the simulation marketplace. Once you learn one simulation language well, it is much easier to switch to other languages and to understand which languages will be more appropriate for certain modeling situations.

Arena<sup>TM</sup> is fundamentally a process-description-based language. That is, when using Arena<sup>TM</sup>, the modeler describes the process that an “entity” experiences while flowing through or using the elements of the system. You will learn about how Arena<sup>TM</sup> facilitates process modeling throughout this textbook.

## 1.7 SIMULATION METHODOLOGY

This section presents a brief overview of the steps of simulation modeling by discussing the process in the context of a methodology. A methodology is simply a series of steps to follow. Since simulation involves systems modeling, a simulation methodology based on the general precepts of solving a problem through systems analysis is presented here. A general methodology for solving problems can be stated as follows:

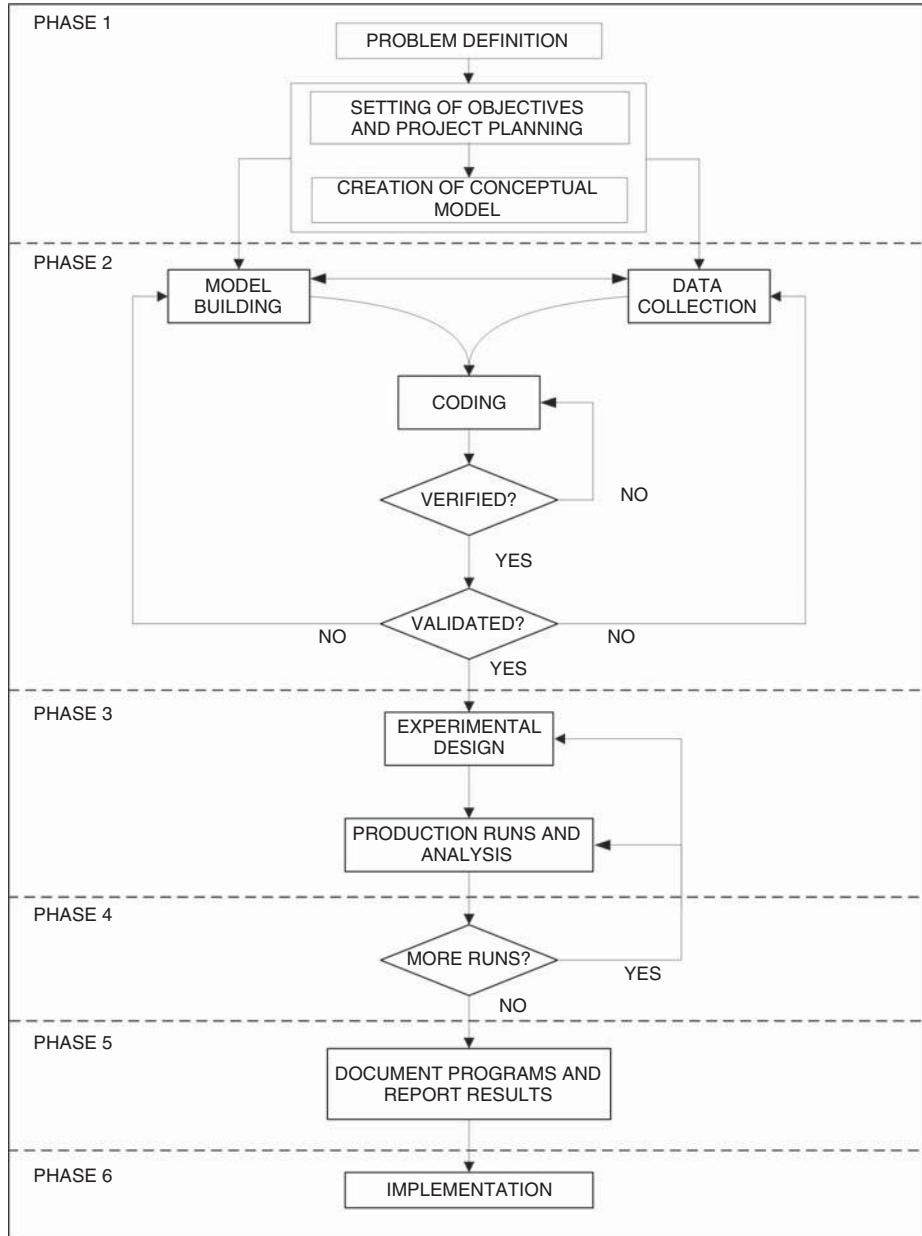
1. Define the problem.
2. Establish measures of performance for evaluation.
3. Generate alternative solutions.
4. Rank alternative solutions.
5. Evaluate and iterate during process.
6. Execute and evaluate the solution.

This methodology can be referred to by using the first letter of each step. The DEGREE methodology for problem solving represents a series of steps that can be used during the problem-solving process. The first step helps to ensure that you are solving the right problem. The second step helps to ensure that you are solving the problem for the right reason, that is, your metrics must be coherent with your problem. Steps 3 and 4 ensure that the analyst looks at and evaluates multiple solutions to the problem. In other words, these steps help to ensure that you develop the right solution to the problem. A good methodology recognizes that the analyst needs to evaluate how well the methodology is doing. In step 5, the analyst evaluates how the process is proceeding and allows for iteration. Iteration is an important concept that is foreign to many modelers. The concept of iteration recognizes that the problem-solving process can be repeated until the desired degree of modeling fidelity has been achieved. Start the modeling at a level that allows it to be initiated and do not try to address the entire situation in each of the steps. Start with small models that work and build them up until you have reached your desired goals. It is important to get started and get something established on each step and continually go back in order to ensure that the model is representing reality in the way that you intended. The final step is often overlooked. Simulation is often used to recommend a solution to a problem. Step 6 indicates that if you have the opportunity, you should execute the solution by implementing the decisions. Finally, you should always follow up to ensure that the projected benefits of the solution were obtained.

The DEGREE problem-solving methodology should serve you well; however, simulation involves certain unique actions that must be performed during the general overall problem-solving process. When applying DEGREE to a problem that may require simulation, the general DEGREE approach needs to be modified to explicitly consider how simulation will interact with the overall problem-solving process.

Figure 1.4 represents a refined general methodology for applying simulation to problem solving.

1. Problem formulation
  - (a) Define the problem
  - (b) Define the system
  - (c) Establish performance metrics
  - (d) Build conceptual model
  - (e) Document model assumptions
2. Simulation model building
  - (a) Model translation
  - (b) Input data modeling
  - (c) Verification
  - (d) Validation



**Figure 1.4** General simulation methodology

3. Experimental design and analysis
  - (a) Preliminary runs
  - (b) Final experiments
  - (c) Analysis of results

4. Evaluate and iterate
  - (a) Documentation
  - (b) Model manual
  - (c) User manual
5. Implementation

The first phase, problem formulation, captures the essence of the first two steps in the DEGREE process. The second phase, model building, captures the essence of step 3 of the DEGREE process. When building models, you are either explicitly or implicitly developing certain design alternatives. The third phase, experimental design and analysis, encapsulates some of steps 3 and 4 of the DEGREE process. In designing experiments, design alternatives are specified, and when analyzing experiments, their worth is being evaluated with respect to problem objectives. The fourth phase, evaluate and iterate, captures the notion of iteration. Finally, the fifth and sixth phases, documentation and implementation complete the simulation process. Documentation is essential when trying to ensure the ongoing and future use of the simulation model, and implementation recognizes that simulation projects often fail if there is no follow through on the recommended solutions.

The problem formulation phase of the study consists of five primary activities:

1. Defining the problem.
2. Defining the system.
3. Establishing performance metrics.
4. Building conceptual models.
5. Documenting modeling assumptions.

A problem starts with a perceived need. These activities are useful in developing an appreciation for and an understanding of what needs to be solved. The basic output of the problem definition activity is a problem definition statement. A problem definition statement is a narrative discussion of the problem. A problem definition statement is necessary to accurately and concisely represent the problem for the analyst and the problem stakeholders. This should include all the required assumptions made during the modeling process. It is important to document your assumptions so that you can examine their effect on the model during the verification, validation, and experimental analysis steps of the methodology. This ensures that the problem is well understood and that all parties agree on the nature of the problem and the goals of the study. The general goals of a simulation study often include the following.

- *Comparison.* To compare system alternatives and their performance measures across various factors (decision variables) with respect to some objectives.
- *Optimization.* This is a special case of comparison in which you try to find the system configuration that optimizes performance subject to constraints.
- *Prediction.* To predict the behavior of the system at some future point in time.
- *Investigation.* To learn about and gain insight into the behavior of the system, given various inputs.

These general goals will need to be specialized to the problem under study. The problem definition should include a detailed description of the objectives of the study, the desired outputs from the model, and the types of scenarios to be examined or decisions to be made.

The second activity of this phase produces a definition of the system. A system definition statement is necessary to accurately and concisely define the system, particularly its boundaries. The system definition statement is a narrative, which often contains a pictorial representation of the major elements of the system. This ensures that the simulation study is focused on the appropriate areas of interest to the stakeholders and that the scope of the project is well understood.

When defining the problem and the system, one should naturally begin to develop an understanding of how to measure system performance. The third activity of problem formulation makes this explicit by encouraging the analyst to define the required performance measures for the model. To meaningfully compare alternative scenarios, objective and measurable metrics describing the performance of the system are necessary. The performance metrics should include quantitative statistical measures from any models used in the analysis (e.g., simulation models), quantitative measures from the systems analysis (e.g., cost/benefits), and qualitative assessments (e.g., technical feasibility, human, operational feasibility). The focus should be placed on the performance measures that are considered to be the most important to system decision makers and tied directly to the objectives of the simulation study. Evaluation of alternatives can then proceed in an objective and unbiased manner to determine which system scenario performs the best according to the decision maker's preferences.

The problem definition statement, the system definition statement, and explicit performance metrics set the stage for more detailed modeling. These activities should be captured in a written form. Within this text, you will develop models of certain "ready-made" book problems. One way to accomplish the problem formulation phase of a simulation study is to consider writing yourself a "book problem." You will need enough detail in these documents that a simulation analyst (you) can develop a model in any simulation language for the given situation. The example problem in Chapter 11 represents an excellent sample of problem and system definition statements. If you have the opportunity to do a "real-life" project as a part of your study of simulation, you might want to utilize the book problems in this text and the example in Chapter 11 for how to write reasonable problem/system definition statements.

With a good understanding of the problem and the system under study, you should be ready to begin your detailed model formulations. Model formulation does not mean a computer program. You should instead use conceptual modeling tools: conceptual diagrams, flow charts, etc. prior to any use of software to implement a model. The purpose of conceptual modeling tools is to convey a more detailed system description so that the model may be translated into a computer representation. General descriptions help to highlight the areas and processes of the system that the model will simulate. Detailed descriptions assist in simulation model development and coding efforts. Some relevant diagramming constructs include the following:

1. *Context Diagrams.* A context diagram assists in conveying the general system description. The diagram is a pictorial representation of the system that often includes flow patterns typically encountered. Context diagrams are often part of the system description document. There are no rules for developing context diagrams. If you have an artistic side, here is your opportunity to shine!

2. *Activity Diagrams.* An activity diagram is a pictorial representation of the process for an entity and its interaction with resources while within the system. If the entity is a temporary entity (i.e., it flows through the system), the activity diagram is called an activity flow diagram. If the entity is permanent (i.e., it remains in the system throughout its life), the activity diagram is called an activity cycle diagram. Activity diagrams will be used extensively within this text.
3. *Software Engineering Diagrams.* Because simulation entails software development, the wide variety of software engineering diagramming techniques can be utilized to provide information for the model builder. Diagrams such as flow charts, database diagrams, IDEF (ICAM Definition language) diagrams, UML (unified modeling language) diagrams, and state charts are all useful in documenting complex modeling situations. These techniques assist development and coding efforts by focusing attention on describing, and thus understanding, the elements within the system. Within this text, activity diagrams will be augmented with some simple flow chart symbols and some simple state diagrams will be used to illustrate a variety of concepts.

In your modeling, you should start with an easy conceptual model that captures the basic aspects and behaviors of the system. Then, you should begin to add details, considering additional functionality. Finally, you should always remember that the complexity of the model has to remain proportional to the quality of the available data and the degree of validity necessary to meet the objectives of the study. In other words, do not try to model the world!

After developing a solid conceptual model of the situation, simulation model building can begin. During the simulation model building phase, alternative system design configurations are developed based on the previously developed conceptual models. Additional project planning is also performed to yield specifications for the equipment, resources, and timing required for the development of the simulation models. The simulation models used to evaluate the alternative solutions are then developed, verified, validated, and prepared for analysis. Within the context of a simulation project, this process includes the following.

- *Input Data Preparation.* Input data is analyzed to determine the nature of the data and further data collection needs. Necessary data is also classified into several areas. This classification establishes different aspects of the model that are used in model development.
- *Model Translation.* Description of the procedure for coding the model, including timing and general procedures and the translation of the conceptual models into computer simulation program representations.
- *Verification.* Verification of the computer simulation model is performed to determine whether or not the program performs as intended. To perform model verification, model debugging is performed to locate any errors in the simulation code. Errors of particular importance include improper flow control or entity creation, failure to release resources, and logical/arithmetic errors or incorrectly observed statistics. Model debugging also includes scenario repetition utilizing identical random number seeds, “stressing” the model through a sensitivity analysis (varying factors and their levels) to ensure compliance with anticipated behavior, and testing of individual modules within the simulation code.
- *Validation.* Validation of the simulation model is performed to determine whether or not the simulation model adequately represents the real system. The simulation model

is shown to personnel (of various levels) associated with the system in question. Their input concerning the realism of the model is critical in establishing the validity of the simulation. In addition, further observations of the system are performed to ensure model validity with respect to actual system performance. A simple technique is to statistically compare the output of the simulation model to the output from the real system and to analyze whether there is a significant (and practical) difference between the two.

Model translation will be a large component of each chapter as you learn how to develop simulation models. Verification and validation techniques will not be a major component of this text, primarily because the models will be examples made for educational purposes. This does not mean that you should ignore this important topic. You are encouraged to examine many of the useful references on validation, see, for example, Balci [1997] and Balci [1998a].

After you are confident that your model has been verified and validated to suit your purposes, you can begin to use the model to perform experiments that investigate the goals and objectives of the project. Preliminary simulation experiments should be performed to set the statistical parameters associated with the main experimental study. The experimental method should use the simulation model to generate benchmark statistics of current system operations. The simulation model is then altered to conform to a potential scenario and is rerun to generate comparative statistics. This process is continued, cycling through suggested scenarios and generating comparative statistics to allow evaluation of alternative solutions. In this manner, objective assessments of alternative scenarios can be made.

For a small set of alternatives, this “one at a time” approach is reasonable; however, often there are a significant number of design factors that can affect the performance of the model. In this situation, the analyst should consider utilizing formal experimental design techniques. This step should include a detailed specification of the experimental design (e.g., factorial) and any advanced output analysis techniques (e.g., batching, initialization bias prevention, variance reduction techniques, and multiple comparison procedures) that may be required during the execution of the experiments. During this step of the process, any quantitative models developed during the previous steps are exercised. Within the context of a simulation project, the computer simulation model is exercised at each of the design points within the stipulated experimental design.

Utilizing the criteria specified by system decision makers, and utilizing the simulation model’s statistical results, alternative scenarios should then be analyzed and ranked. A methodology should be used to allow the comparison of the scenarios that have multiple performance measures that trade-off against each other.

If you are satisfied that the simulation has achieved your objectives, then you should document and implement the recommended solutions. If not, you can iterate as necessary and determine if any additional data, models, experimentation, or analysis is needed to achieve your modeling objectives. Good documentation should consist of at least two parts: a technical manual, which can be used by the same analyst or by other analysts, and a user manual. A good technical manual is very useful when the project has to be modified, and it can be a very important contribution to software reusability and portability. The approach to document the example models in this text can be used as an example for how to document your models. In addition to good model development documentation, often the simulation model will be used by non-analysts. In this situation, a good user manual for how to use and exercise the model is imperative. The user manual is a product for the user who may

not be an expert in programming or simulation issues; therefore, clearness and simplicity should be its main characteristics. If within the scope of the project, the analyst should also develop implementation plans and follow through with the installation and integration of the proposed solutions. After implementation, the project should be evaluated as to whether or not the proposed solution met the intended objectives.

## 1.8 ORGANIZATION OF THE BOOK

This chapter introduced some of the basic concepts in simulation. Chapter 2 presents the mathematical basis for random number generation and for generating random variables from probability distributions. Chapter 3 will expand on the topic of modeling randomness within simulation within a spreadsheet environment. Chapter 4 will begin our exploration of discrete-event simulation and introduce you to the key modeling tool, Arena™. Chapter 5 present the details of process modeling within Arena™. After these five chapters, you should be able to model a variety of interesting systems. Good simulation models are based on modeling the data that drives the simulation with probability models. Chapter 6 describes the basic processes for modeling simulation input distributions. Whenever a simulation utilizes probability models for input parameters, it will also produce output that is random. Chapter 7 examines some of the statistical issues involved in analyzing simulation output data. You will also see how Arena™ makes this analysis easier for the user. These first seven chapters will enable you to build and analyze models using appropriate statistical methods.

The rest of the book focuses on building deeper modeling skills. Chapter 8 then examines some common modeling situation involving queues and inventory systems. The modeling of situations involving material handling systems is presented in Chapter 9. Then Chapter 10 discusses some miscellaneous topics in simulation modeling and examines some of the programming capabilities of Arena™. The final chapter illustrates the use of simulation on a practical case study. Through this study you will have a solid foundation for understanding what it takes to model more realistic systems found in practice.

Simulation is a tool that can assist analysts in improving system performance. There are many other aspects of simulation besides Arena™ that will be considered within this text. I hope that you will find this a useful and interesting experience.

## EXERCISES

- 1.1 Using the resources at <http://www.informs-sim.org/wscpapers.html>, find an application of simulation to a real system and discuss why simulation was important to the analysis.
- 1.2 Read the paper: Balci [1998b]. (<http://www.informs-cs.org/wsc98papers/006.PDF>) and discuss the difference between verification and validation. Why is the verification and validation of a simulation model important?
- 1.3 Customers arrive to a gas station with two pumps. Each pump can reasonably accommodate a total of two cars. If all the space for the cars is full, potential customers will balk (leave without getting gas). What measures of performance will be useful in evaluating the effectiveness of the gas station? Describe how you would collect the interarrival and service times of the customers necessary to simulate this system.

- 1.4 Classify the systems as being either discrete or continuous:
- (a) Electrical capacitor (You are interested in modeling the amount of current in a capacitor at any time  $t$ ).
  - (b) On-line gaming system. (You are interested in modeling the number of people playing Halo 4 at any time  $t$ .)
  - (c) An airport. (You are interested in modeling the percentage of flights that depart late on any given day).
- 1.5 Classify the systems as being either discrete or continuous:
- (a) Parking lot
  - (b) Level of gas in Fayetteville shale deposit
  - (c) Printed circuit board manufacturing facility
- 1.6 Classify the systems as being either discrete or continuous:
- (a) Elevator system (You are interested in modeling the number of people waiting on each floor and traveling within the elevators.)
  - (b) Judicial system (You are interested in modeling the number of cases waiting for trial.)
  - (c) The in-air flight path of an airplane as it moves from an origin to a destination.
- 1.7 What is model conceptualization? Give an example of something that might be produced during model conceptualization.
- 1.8 The act of implementing the model in computer code, including timing and general procedures and the representation of the conceptual model into a computer simulation program is called \_\_\_\_\_.
- 1.9 Which of the following does the problem formulation phase of simulation not include?
- (a) Define the system
  - (b) Establish performance metrics
  - (c) Verification
  - (d) Build conceptual models
- 1.10 The general goals of a simulation study often include (a)\_\_\_\_\_ of system alternatives and their performance measures across various factors (decision variables) with respect to some objectives. (b)\_\_\_\_\_ of system behavior at some future point in time.
- 1.11 *True or False* Verification of the simulation model is performed to determine whether the simulation model adequately represents the real system.

---

# 2

---

## GENERATING RANDOMNESS IN SIMULATION

### LEARNING OBJECTIVES

- To be able to describe and use linear congruential pseudorandom number generation methods
- To be aware of current state-of-the-art pseudorandom number generation methods.
- To be able to define and use key terms in pseudorandom number generation methods such as streams, seeds, and period.
- To be able to explain the key issues in pseudorandom number testing.
- To be able to derive and implement an inverse cumulative distribution function (CDF)-based random variate generation algorithm.
- To be able to explain and implement the convolution algorithm for random variate generation.
- To be able to explain and implement the acceptance rejection algorithm for random variate generation.

### 2.1 THE STOCHASTIC NATURE OF SIMULATION

One of the most important advantages of using simulation models to characterize system performance is the flexibility that simulation allows for incorporating randomness into the modeling. As discussed briefly in Chapter 1, randomness in simulation is often modeled by using random variables and probability distributions. Thus, simulation languages require

the ability to generate random variates. A random variate is an instance (or realization) of a random variable.

In this chapter, you will learn how simulation languages allow for the generation of randomness. Generating randomness requires algorithms that permit sequences of numbers to act as the underlying source of randomness within the model. A basic understanding of these algorithms will be the focus of the first part of the chapter. Then, given a good source of randomness, techniques that permit the sequences to be transformed so that they can represent a wide variety of random variables (e.g., normal and Poisson) have been established. The algorithms that govern these procedures are described in the second part of the chapter.

## 2.2 RANDOM NUMBERS

This section will indicate how uniformly distributed random numbers over the range from 0 to 1 are obtained within simulation programs. While commercial simulation packages provide substantial capabilities for generating random numbers, we still need to understand how this process works for the following reasons:

1. The random numbers within a simulation experiment might need to be controlled in order to take advantage of them to improve decision making.
2. In some situations, the commercial package does not have ready-made functions for generating the desired random variables. In these situations, you will have to do it yourself.

In addition, simulation is much broader than just using a commercial package. You can perform simulation in any computer language and the informed modeler should know how the key inputs to simulations are achieved.

In simulation, large amount of cheap (easily computed) random numbers are required. In general, consider how random numbers might be obtained:

1. Dice, coins, colored balls
2. Specially designed electronic equipment
3. Algorithms.

Clearly, within the context of computer simulation, it might be best to rely on algorithms; however, if an algorithm is used to generate the random numbers, then they cannot be truly random. For this reason, the random numbers that are used in computer simulation are called *pseudorandom*.

**Definition 2.1 (Pseudorandom Numbers)** *A sequence of pseudorandom numbers,  $U_i$ , is a deterministic sequence of numbers in  $(0, 1)$  having the same relevant statistical properties as a sequence of truly random  $U(0, 1)$  numbers. (Ripley [1987])*

A set of statistical tests are performed on the pseudorandom numbers generated from algorithms in order to indicate that their properties are not significantly different from a true set of  $U(0, 1)$  random numbers. The algorithms that produce pseudorandom numbers are called random number generators. In addition to passing a battery of tests, the random

number generators need to be fast and they need to be able to reproduce a sequence of numbers if and when necessary.

The following section discusses random number generation methods. The approach will be practical, with just enough theory to motivate future study of this area and to allow you to understand the important implications of random number generation. A more rigorous treatment of random number and random variable generation can be found in texts such as Fishman [2006] and Devroye [1986].

### 2.3 RANDOM NUMBER GENERATORS

Over the history of scientific computing, there have been a wide variety of techniques and algorithms proposed and used for generating pseudorandom numbers. A common technique that has been used (and is still in use) within a number of simulation environments is discussed in this chapter. Some new types of generators that have been recently adopted within many simulation environments, especially the one used within Arena<sup>TM</sup>, will also be briefly discussed.

A linear congruential generator (LCG) is a recursive algorithm for producing a sequence of pseudorandom numbers. Each new pseudorandom number from the algorithm depends on the previous pseudorandom number. Thus, a starting value called the *seed* is required. Given the value of the seed, the rest of the sequence of pseudorandom numbers can be completely determined by the algorithm. The basic definition of an LCG is as follows:

**Definition 2.2 (LCG)** *An LCG defines a sequence of integers,  $R_0, R_1, \dots$  between 0 and  $m - 1$  according to the following recursive relationship, where  $i = 0, 1, 2, \dots$ :*

$$R_{i+1} = (aR_i + c) \bmod m \quad \text{for } i = 0, 1, 2, \dots \quad (2.1)$$

where  $R_0$  is called the *seed* of the sequence,  $a$  is called the *constant multiplier*,  $c$  is called the *increment*, and  $m$  is called the *modulus*. ( $m, a, c, R_0$ ) are integers with  $a > 0$ ,  $c \geq 0$ ,  $m > a$ ,  $m > c$ ,  $m > R_0$ , and  $0 \leq R_i \leq m - 1$ .

To compute the corresponding pseudorandom uniform number, we use

$$U_i = \frac{R_i}{m} \quad (2.2)$$

Notice that an LCG defines a sequence of integers and subsequently a sequence of real (rational) numbers that can be considered pseudorandom numbers. Remember that pseudorandom numbers are those that can “fool” a battery of statistical tests. The choice of the seed, constant multiplier, increment, and modulus, that is, the parameters of the LCG, will determine the properties of the sequences produced by the generator. With properly chosen parameters, an LCG can be made to produce pseudorandom numbers. To make this concrete, let us look at a simple example of an LCG.

#### ■ EXAMPLE 2.1 Simple LCG

Consider an LCG with parameters ( $m = 8, a = 5, c = 1, R_0 = 5$ ). Compute the first nine values for  $R_i$  and  $U_i$  from the defined sequence.

Let us first remember how to compute using the `mod` operator. The `mod` operator is defined as

$$\begin{aligned} z &= y \bmod m \\ &= y - m \left\lfloor \frac{y}{m} \right\rfloor \end{aligned}$$

where  $\lfloor \cdot \rfloor$  is the floor operator, which returns the greatest integer that is less than or equal to  $x$ . For example,

$$\begin{aligned} z &= 17 \bmod 3 \\ &= 17 - 3 \left\lfloor \frac{17}{3} \right\rfloor \\ &= 17 - \lfloor 5.66 \rfloor \\ &= 17 - 3 \times 5 = 2 \end{aligned}$$

Thus, the `mod` operator returns the integer remainder (including zero) when  $y \geq m$  and  $y$  when  $y < m$ . For example,  $z = 6 \bmod 9 = 6 - 9 \lfloor \frac{6}{9} \rfloor = 6 - 9 \times 0 = 6$ . Using the parameters of the LCG in Equation (2.1), the pseudorandom numbers are

$$\begin{aligned} R_1 &= (5R_0 + 1) \bmod 8 = 26 \bmod 8 = 2 \Rightarrow U_1 = 0.25 \\ R_2 &= (5R_1 + 1) \bmod 8 = 11 \bmod 8 = 3 \Rightarrow U_2 = 0.375 \\ R_3 &= (5R_2 + 1) \bmod 8 = 16 \bmod 8 = 0 \Rightarrow U_3 = 0.0 \\ R_4 &= (5R_3 + 1) \bmod 8 = 1 \bmod 8 = 1 \Rightarrow U_4 = 0.125 \\ R_5 &= 6 \Rightarrow U_5 = 0.75 \\ R_6 &= 7 \Rightarrow U_6 = 0.875 \\ R_7 &= 4 \Rightarrow U_7 = 0.5 \\ R_8 &= 5 \Rightarrow U_8 = 0.625 \\ R_9 &= 2 \Rightarrow U_9 = 0.25 \end{aligned}$$

In Example 2.1, the  $U_i$  are simple fractions involving  $m = 8$ . Certainly, this sequence does not appear very random. The  $U_i$  can only take on rational values in the range,  $0, \frac{1}{m}, \frac{2}{m}, \frac{3}{m}, \dots, \frac{(m-1)}{m}$  since  $0 \leq R_i \leq m - 1$ . This implies that if  $m$  is small, there will be gaps on the interval  $[0, 1)$  and if  $m$  is large, then the  $U_i$  will be more densely distributed on  $[0, 1)$ .

Notice that if a sequence generates the same value as a previously generated value, then the sequence will repeat or cycle. An important property of an LCG is that it has a long cycle, as close to length  $m$  as possible. The length of the cycle is called the *period* of the LCG. Ideally, the period of the LCG is equal to  $m$ . If this occurs, the LCG is said to achieve its full period. As can be seen in the example, the LCG is full period. Until recently, most computers were 32-bit machines and thus a common value for  $m$  is  $2^{31} - 1 = 2,147,483,647$ , which

represents the largest integer number on a 32-bit computer using 2's complement integer arithmetic. This choice of  $m$  also happens to be a prime number, which leads to special properties.

A proper choice of the parameters of the LCG will allow desirable pseudorandom number properties to be obtained. The following result due to Hull and Dobell [1962], see also Law [2007], indicates how to check if an LCG will have the largest possible cycle.

**Theorem 2.1 (LCG Full Period Conditions)** *An LCG has full period if and only if the following three conditions hold:*

1. *The only positive integer that (exactly) divides both  $m$  and  $c$  is 1 (i.e.,  $c$  and  $m$  have no common factors other than 1).*
2. *If  $q$  is a prime number that divides  $m$  then  $q$  should divide  $(a - 1)$  (i.e.,  $(a - 1)$  is a multiple of every prime number that divides  $m$ ).*
3. *If 4 divides  $m$ , then 4 should divide  $(a - 1)$  (i.e.,  $(a - 1)$  is a multiple of 4 if  $m$  is a multiple of 4).*

Now, let us apply this theorem to the example LCG and check whether or not it should obtain full period.

### ■ EXAMPLE 2.2 Checking if LCG Reaches Full Period

Use Theorem 2.1 to check if the LCG of Example 2.1 should reach its full period.

To apply the theorem, you must check if each of the three conditions holds for the generator.

- Condition 1:  $c$  and  $m$  have no common factors other than 1.  
The factors of  $m = 8$  are  $(1, 2, 4, 8)$ , since  $c = 1$  (with factor 1) condition 1 is true.
- Condition 2:  $(a - 1)$  is a multiple of every prime number that divides  $m$ . The first few prime numbers are  $(1, 2, 3, 5, 7)$ . The prime numbers,  $q$ , that divide  $m = 8$  are  $(q = 1, 2)$ . Since  $a = 5$  and  $(a - 1) = 4$ , clearly  $q = 1$  divides 4 and  $q = 2$  divides 4. Thus, condition 2 is true.
- Condition 3: If 4 divides  $m$ , then 4 should divide  $(a - 1)$ .  
Since  $m = 8$ , clearly 4 divides  $m$ . Also, 4 divides  $(a - 1) = 4$ . Thus, condition 3 holds.

Since all three conditions hold, the LCG achieves full period.

There are some simplifying conditions, see Banks et al. [2005], which allow for easier application of the theorem. For  $m = 2^b$  ( $m$  a power of 2) and  $c$  not equal to 0, the longest possible period is  $m$  and can be achieved, provided that  $c$  is chosen so that the greatest common factor of  $c$  and  $m$  is 1 and  $a = 4k + 1$  where  $k$  is an integer. The previous example LCG satisfies this situation.

For  $m = 2^b$  and  $c = 0$ , the longest possible period is  $(m/4)$  and can be achieved, provided that the initial seed,  $R_0$  is odd and  $a = 8k + 3$  or  $a = 8k + 5$  where  $k = 0, 1, 2, \dots$ .

The case in which  $m$  is a prime number and  $c = 0$  defines a special case of the LCG called a prime modulus multiplicative linear congruential generator (PMLLCG).

For this case, the longest possible period is  $m - 1$  and can be achieved if the smallest integer,  $k$ , such that  $a^k - 1$  is divisible by  $m$  is  $m - 1$ .

Thirty-Two-bit computers have been very common for over 20 years. In addition,  $2^{31} - 1 = 2,147,483,647$  is a prime number. Because of this,  $2^{31} - 1$  has been the choice for  $m$  with  $c = 0$ . Two common values for the multiplier,  $a$ , have been

$$a = 630,360,016$$

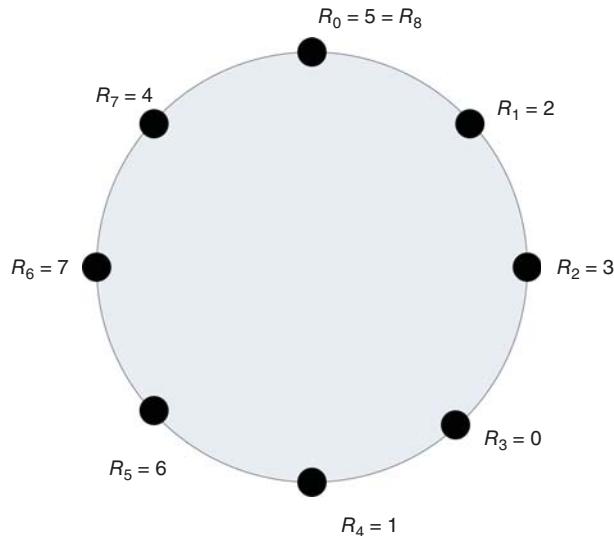
$$a = 16,807$$

The latter of which was used within Arena™ for a number of years. Notice that for PMMLCG, the full period cannot be achieved (because  $c = 0$ ), but with the proper selection of the multiplier, the next best period length of  $m - 1$  can be obtained. In addition, for this case,  $R_0 \in \{1, 2, \dots, m - 1\}$  and thus  $U_i \in (0, 1)$ . The limitation of  $U_i \in (0, 1)$  is very useful when generating random variables from various probability distributions, since 0 cannot be realized.

When using an LCG, you must supply a starting seed as an initial value for the algorithm. This seed determines the sequence that will come out of the generator when it is called within software. Since generators cycle, you can think of the sequence as a big circular list as indicated in Figure 2.1. Starting with seed  $R_0 = 5$ , you get a sequence  $\{2, 3, 0, 1, 6, 7, 4, 5\}$ . Starting with seed,  $R_0 = 1$ , you get the sequence  $\{6, 7, 4, 5, 2, 3, 0, 1\}$ . Notice that these two sequences overlap with each other, but that the first half  $\{2, 3, 0, 1\}$  and the second half  $\{6, 7, 4, 5\}$  of the sequence do not overlap. If you only use 4 random numbers from each of these two *subsequences*, then the numbers will not overlap. This leads to the definition of a stream:

**Definition 2.3 (Random Number Stream)** *The subsequence of random numbers generated from a given seed is called a random number stream.*

You can take the sequence produced by the random number generator and divide it up into subsequences by associating certain seeds with streams. You can call the first



**Figure 2.1**  $R_i$  sequence for Example 2.1.

subsequence stream 1 and the second subsequence stream 2, and so forth. Each stream can be further divided into subsequences or substreams of nonoverlapping random numbers.

In this simple example, it is easy to remember that stream 1 is defined by seed,  $R_0 = 5$ , but when  $m$  is large, the seeds will be large integer numbers, for example,  $R_0 = 123098345$ . It is difficult to remember such large numbers. Rather than remembering this huge integer, an assignment of stream numbers to seeds is made. Then, the sequence can be reference by its stream number. Naturally, if you are going to associate seeds with streams, you would want to divide the entire sequence so that the number of nonoverlapping random numbers in each stream is quite large. This ensures that as a particular stream is used that there is very little chance of continuing into the next stream. Clearly, you want  $m$  to be as large as possible and to have many streams that contain as large as possible number of nonoverlapping random numbers. With today's modern computers, even  $m$  is  $2^{31} - 1 = 2,147,483,647$  is not very big. For large simulations, you can easily run through all these random numbers.

Random number generators in computer simulation languages come with a default set of streams that divide the "circle" up into independent sets of random numbers. The streams are only independent if you do not use up all the random numbers within the subsequence. These streams allow the randomness associated with a simulation to be controlled. During the simulation, you can associate a specific stream with specific random processes in the model. This has the advantage of allowing you to check if the random numbers are causing significant differences in the outputs. In addition, this allows the random numbers used across alternative simulations to be better synchronized.

Now a common question for beginners using random number generators can be answered. That is, *If the simulation is using random numbers, why do I get the same results each time I run my program?* The corollary to this question is, *If I want to get different random results each time I run my program, how do I do it?* The answer to the first question is that the underlying random number generator is starting with the same seed each time you run your program. Thus, your program will use the same pseudorandom numbers today as it did yesterday and the day before, etc. The answer to the corollary question is that you must tell the random number generator to use a different seed (or, alternatively, a different stream) if you want different invocations of the program to produce different results. The latter is not necessarily a desirable goal. For example, when developing your simulation programs, it is desirable to have repeatable results so that you can know that your program is working correctly. Unfortunately, many novices have heard about using the computer clock to "randomly" set the seed for a simulation program. This is a *bad* idea and very much not recommended in our context. This idea is more appropriate within a gaming simulation, in order to allow the human gamer to experience different random sequences.

Given current computing power, the previously discussed PMMLCGs are insufficient since it is likely that all the 2 billion or so of the random numbers would be used in performing serious simulation studies. Thus, a new class of random number generators that have extremely long periods was developed. The random number generator described in L'Ecuyer et al. [2002] is one example of such a generator. It is based on the combination of two multiple recursive generators resulting in a period of approximately  $3.1 \times 10^{57}$ . This is the same generator that is now used in many commercial simulation packages. The generator as defined in Law [2007] is

$$R_{1,i} = (1,403,580R_{1,i-2} - 810,728R_{1,i-3})[\text{mod}(2^{32} - 209)] \quad (2.3)$$

$$R_{2,i} = (527,612R_{2,i-1} - 1,370,589R_{2,i-3})[\text{mod}(2^{32} - 22,853)] \quad (2.4)$$

$$Y_i = (R_{1,i} - R_{2,i})[\text{mod}(2^{32} - 209)] \quad (2.5)$$

$$U_i = \frac{Y_i}{2^{32} - 209} \quad (2.6)$$

To illustrate how this generator works, consider generating an initial sequence of pseudorandom numbers from the generator. The generator takes as its initial seed a vector of six initial values  $(R_{1,0}, R_{1,1}, R_{1,2}, R_{2,0}, R_{2,1}, R_{2,2})$ . The first initially generated value,  $U_i$ , will start at index 3. Example 2.3 illustrates the recursion to generate five pseudorandom numbers.

### EXAMPLE 2.3 L'Ecuyer et al. (2002) Generator

Produce five pseudorandom numbers from the L'Ecuyer et al. [2002] generator using the initial seed vector:

$$\{R_{1,0}, R_{1,1}, R_{1,2}, R_{2,0}, R_{2,1}, R_{2,2}\} = \{12345, 12345, 12345, 12345, 12345, 12345\}$$

	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$
$Z_{1,i-3} =$	12345	12345	12345	3023790853	3023790853
$Z_{1,i-2} =$	12345	12345	3023790853	3023790853	3385359573
$Z_{1,i-1} =$	12345	3023790853	3023790853	3385359573	1322208174
$Z_{2,i-3} =$	12345	12345	12345	2478282264	1655725443
$Z_{2,i-2} =$	12345	12345	2478282264	1655725443	2057415812
$Z_{2,i-1} =$	12345	2478282264	1655725443	2057415812	2070190165
$Z_{1,i} =$	3023790853	3023790853	3385359573	1322208174	2930192941
$Z_{2,i} =$	2478282264	1655725443	2057415812	2070190165	1978299747
$Y_i =$	545508589	1368065410	1327943761	3546985096	951893194
$U_i =$	0.127011122076	0.318527565471	0.309186015655	0.82584686312	0.221629915834

While it is beyond the scope of this text to explore the theoretical underpinnings of this generator, it is important to note that the use of this new generator is conceptually similar to that which has already been described. The generator allows multiple independent streams to be defined along with substreams.

The fantastic thing about this generator is the sheer size of the period. Based on their analysis, L'Ecuyer et al. [2002] stated that it will be “approximately 219 years into the future before average desktop computers will have the capability to exhaust the cycle of the (generator) in a year of continuous computing.” In addition to the period length, the generator has an enormous number of streams, approximately  $1.8 \times 10^{19}$  with stream lengths of  $1.7 \times 10^{38}$  and substreams of length  $7.6 \times 10^{22}$  numbering at  $2.3 \times 10^{15}$  per stream. Clearly, with these properties, you do not have to worry about overlapping random numbers when performing simulation experiments. The generator was subjected to a rigorous battery of statistical tests and is known to have excellent statistical properties.

## 2.4 TESTING RANDOM NUMBERS

This subsection closes with an overview of what is involved in testing the statistical properties of random number generators. Essentially a random number generator is supposed to produce sequences of numbers that appear to be independent and identically distributed

(IID)  $U(0, 1)$  random variables. The hypothesis that a sample from the generator is IID  $U(0, 1)$  must be made. Then, the hypothesis is subjected to various statistical tests. There are standard batteries of test, see, for example, Soto [1999] and L'Ecuyer [1992], that are utilized. Typical tests examine the following.

- Distributional properties: for example, chi-squared and Kolmogorov–Smirnov (K-S) test
- Independence: for example, correlation tests and runs tests
- Patterns: for example, Poker test and gap test

When considering the quality of random number generators, the higher dimensional properties of the sequence also need to be considered. For example, the serial test is a higher dimensional chi-squared test.

#### 2.4.1 Distributional Tests

In essence, a distributional test examines the hypothesis  $H_0 : U_i \sim U(0, 1)$  versus the alternate hypothesis of  $H_a : U_i \not\sim U(0, 1)$ . That is, the null hypothesis is that the generated numbers are uniformly distributed on the interval 0 to 1 and the alternative hypothesis that the generated numbers are not uniformly distributed on the interval 0 to 1.

As a reminder, when using a hypothesis testing framework, we can perform the test in two ways: (i) prespecifying the Type 1 error and comparing to a critical value or (ii) prespecifying the Type 1 error and comparing to a  $p$ -value. For example, in the first case, suppose we let our Type 1 error  $\alpha = 0.05$ , then we look up a critical value appropriate for the test, compute the test statistic, and reject the null hypothesis  $H_0$  if test statistic is too extreme (large or small depending on the test). In the second case, we compute the  $p$ -value for the test based on the computed test statistic and then reject  $H_0$  if the  $p$ -value is less than or equal to the specified Type 1 error value.

This text emphasizes the  $p$ -value approach to hypothesis testing because computing the  $p$ -value provides additional information about the significance of the result. The  $p$ -value for a statistical test is the smallest  $\alpha$  level at which the observed test statistic is significant. This smallest  $\alpha$  level is also called the observed level of significance for the test. The smaller the  $p$ -value, the more the result can be considered statistically significant. Thus, the  $p$ -value can be compared to the desired significance level,  $\alpha$ . Thus, when using the  $p$ -value approach to hypothesis testing, the testing criterion is

- If the  $p$ -value  $> \alpha$ , then do not reject  $H_0$
- If the  $p$ -value  $\leq \alpha$ , then reject  $H_0$ .

An alternate interpretation for the  $p$ -value is that it is the probability assuming  $H_0$  is true of obtaining a test statistic at least as *extreme* as the observed value. Assuming that  $H_0$  is true, then the test statistic will follow a known distributional model. The  $p$ -value can be interpreted as the chance assuming that  $H_0$  is true of obtaining a more extreme value of the test statistic than the value actually obtained. Computing a  $p$ -value for a hypothesis test allows for a different mechanism for accepting or rejecting the null hypothesis. Remember that a Type I error is

$$\alpha = P(\text{Type 1 error}) = P(\text{rejecting the null when it is in fact true})$$

This represents the chance you are willing to take to make a mistake in your conclusion. The  $p$ -value represents the observed chance associated with the test statistic being more extreme under the assumption that the null is true. A small  $p$ -value indicates that the observed result is rare under the assumption of  $H_0$ . If the  $p$ -value is small, it indicates that an outcome as extreme as observed is possible, but not probable under  $H_0$ . In other words, chance by itself may not adequately explain the result. So for a small  $p$ -value, you can reject  $H_0$  as a plausible explanation of the observed outcome. If the  $p$ -value is large, this indicates that an outcome as extreme as that observed can occur with high probability. For example, suppose you observed a  $p$ -value of 0.001. This means that assuming  $H_0$  is true, there was a 1 in 1000 chance that you would observe a test statistic value at least as extreme as what you observed. It comes down to whether or not you consider a 1 in 1000 occurrence a rare or non-rare event. In other words, do you consider yourself that lucky to have observed such a rare event. If you do not think of this as lucky but you still actually observed it, then you might be suspicious that something is not “right with the world.” In other words, your assumption that  $H_0$  was true is likely wrong. Therefore, you reject the null hypothesis,  $H_0$ , in favor of the alternative hypothesis,  $H_a$ . The risk of you making an incorrect decision here is what you consider a rare event, that is, your  $\alpha$  level.

There are two major tests that are utilized to examine the distributional properties of sequences of pseudorandom numbers: chi-squared goodness-of-fit test and the K-S test.

**2.4.1.1 Chi-Squared Goodness-of-Fit Test** The chi-square test divides the range of the data into,  $k$ , intervals and tests if the number of observations that fall in each interval is close the expected number that should fall in the interval, given the hypothesized distribution is the correct model. Let  $\vec{x} = (x_1, x_2, \dots, x_n)$  represent the set of observations. Let  $b_0, b_1, \dots, b_k$  be the breakpoints (end points) of the class intervals such that  $(b_0, b_1], (b_1, b_2], \dots, (b_{k-1}, b_k]$  form  $k$  disjoint and adjacent intervals. The intervals do not have to be of equal width. Also,  $b_0$  can be equal to  $-\infty$  resulting in interval  $(-\infty, b_1]$  and  $b_k$  can be equal to  $+\infty$  resulting in interval  $(b_{k-1}, +\infty)$ .

To count the number of observations that fall in each interval, it is useful to define the following function:

$$c(\vec{x} \leq b) = \#\{x_i \leq b\} \quad i = 1, \dots, n \quad (2.7)$$

$c(\vec{x} \leq b)$  counts the number of observations less than or equal to  $x$ . Let  $c_j$  be the observed count of the  $x$  values contained in the  $j$ th interval  $(b_{j-1}, b_j]$ . Then, we can determine  $c_j$  via the following equation:

$$c_j = c(\vec{x} \leq b_j) - c(\vec{x} \leq b_{j-1}) \quad (2.8)$$

Note that  $\sum_{j=1}^k c_j = n$ . The COUNTIF(cell range, criteria) function in your favorite spreadsheet application can readily implement Equations (2.7) and (2.8). The chi-squared test statistic has the form

$$\chi_0^2 = \sum_{j=1}^k \frac{(c_j - np_j)^2}{np_j} \quad (2.9)$$

The quantity  $np_j$  is the expected number of observations that should fall in the  $j$ th interval when there are  $n$  observations.

For large  $n$ , an approximate  $1 - \alpha$  level hypothesis test can be performed based on a chi-squared test statistic that rejects the null hypothesis if the computed  $\chi_0^2$  is greater than

$\chi^2_{\alpha,k-s-1}$ , where  $s$  is the number of estimated parameters for the hypothesized distribution.  $\chi^2_{\alpha,k-s-1}$  is the upper critical value for a chi-squared random variable  $\chi^2$  such that  $P\{\chi^2 > \chi^2_{\alpha,k-s-1}\} = \alpha$ . The  $p$ -value,  $P\{\chi^2_{k-s-1} > \chi^2_0\}$ , for this test can be computed in Excel<sup>TM</sup> using the following formula:

$$\text{CHISQ.DIST.RT}(\chi^2_0, k - s - 1)$$

In the statistical package  $R$ , the formula is

$$\text{pchisq}(\chi^2_0, k - s - 1, lower.tail = FALSE)$$

The null hypothesis is that the data come from the hypothesized distribution versus the alternative hypothesis that the data do not come from the hypothesized distribution.

For testing if the data are  $U(0, 1)$ , the following is the typical procedure:

1. Divide the interval  $(0, 1)$  into  $k$  equally spaced classes so that  $\Delta b = b_j - b_{j-1}$  resulting in  $p_j = \frac{1}{k}$  for  $j = 1, 2, \dots, k$ . This results in the expected number in each interval being  $np_j = n \times \frac{1}{k} = \frac{n}{k}$ .
2. As a practical rule, the expected number in each interval  $np_j$  should be at least 5. Thus, in this case,  $\frac{n}{k} \geq 5$  or  $n \geq 5k$  or  $k \leq \frac{n}{5}$ . Thus, for a given value of  $n$ , you should choose the number of intervals  $k \leq \frac{n}{5}$ .
3. Since the parameters of the distribution are known  $a = 0$  and  $b = 1$ , then  $s = 0$ . Therefore, we reject  $H_0 : U_i \sim U(0, 1)$  if  $\chi^2_0 > \chi^2_{\alpha,k-1}$  or if the  $p$ -value is less than  $\alpha$ .

If  $p_j$  is chosen as  $\frac{1}{k}$ , then Equation (2.9) can be rewritten as

$$\chi^2_0 = \frac{k}{n} \sum_{j=1}^k \left( c_j - \frac{n}{k} \right)^2 \quad (2.10)$$

## ■ EXAMPLE 2.4 Chi-squared Test on Pseudorandom Numbers

Suppose we have 100 observations from a pseudorandom number generator. Perform a chi-squared test that the numbers appear  $U(0, 1)$ .

0.971	0.668	0.742	0.171	0.350	0.931	0.803	0.848	0.160	0.085
0.687	0.799	0.530	0.933	0.105	0.783	0.828	0.177	0.535	0.601
0.314	0.345	0.034	0.472	0.607	0.501	0.818	0.506	0.407	0.675
0.752	0.771	0.006	0.749	0.116	0.849	0.016	0.605	0.920	0.856
0.830	0.746	0.531	0.686	0.254	0.139	0.911	0.493	0.684	0.938
0.040	0.798	0.845	0.461	0.385	0.099	0.724	0.636	0.846	0.897
0.468	0.339	0.079	0.902	0.866	0.054	0.265	0.586	0.638	0.869
0.951	0.842	0.241	0.251	0.548	0.952	0.017	0.544	0.316	0.710
0.074	0.730	0.285	0.940	0.214	0.679	0.087	0.700	0.332	0.610
0.061	0.164	0.775	0.015	0.224	0.474	0.521	0.777	0.764	0.144

Since we have  $n = 100$ , the number of intervals should be less than or equal to 20. Let us choose  $k = 10$ . This means that  $p_j = 0.1$  for all  $j$ .

$j$	$b_{j-1}$	$b_j$	$p_j$	$c(\vec{x} \leq b_{j-1})$	$c(\vec{x} \leq b_j)$	$c_j$	$np_j$	$\frac{(c_j - np_j)^2}{np_j}$
1	0	0.1	0.1	0	13	13	10	0.9
2	0.1	0.2	0.1	13	21	8	10	0.4
3	0.2	0.3	0.1	21	28	7	10	0.9
4	0.3	0.4	0.1	28	35	7	10	0.9
5	0.4	0.5	0.1	35	41	6	10	1.6
6	0.5	0.6	0.1	41	50	9	10	0.1
7	0.6	0.7	0.1	50	63	13	10	0.9
8	0.7	0.8	0.1	63	77	14	10	1.6
9	0.8	0.9	0.1	77	90	13	10	0.9
10	0.9	1	0.1	90	100	10	10	0

Summing the last column yields

$$\chi_0^2 = \sum_{j=1}^k \frac{(c_j - np_j)^2}{np_j} = 8.2$$

Computing the  $p$ -value for  $k - s - 1 = 10 - 0 - 1 = 9$  degrees of freedom yields  $P\{\chi_9^2 > 8.2\} = 0.514$ . Thus, given such a high  $p$ -value, we would not reject the hypothesis that the observed data is  $U(0, 1)$ .

A spreadsheet implementation for Example 2.4 is available on the *Chi-square Test U(0,1)* tab of the spreadsheet *Ch2SpreadsheetExamples* that accompanies this chapter. In addition, the test can be readily performed using the R statistical software package. Assuming that the file *u01data.txt* contains the PRNs for this example, then the following R commands will perform the test:

```
> data <- read.table("u01data.txt")
> b = seq(0,1, by = 0.1)
> h = hist(data$V1, b, right = FALSE)
> chisq.test(h$counts)

Chi-squared test for given probabilities

data: h$counts
X-squared = 8.2, df = 9, p-value = 0.5141
```

**2.4.1.2 Higher Dimensional Chi-Squared Test** The pseudorandom numbers should not only be uniformly distributed on the interval  $(0,1)$  but they should also be uniformly distributed within the unit square,  $\{(x, y) : x \in (0, 1), y \in (0, 1)\}$ , the unit cube, and so forth for higher number of dimensions  $d$ .

The serial test, described in Law (2007), can be used to assess the quality of the higher dimensional properties of pseudorandom numbers. Suppose the sequence of pseudorandom numbers can be formed into nonoverlapping vectors each of size  $d$ . The vectors should be IID random vectors uniformly distributed on the  $d$ -dimensional unit hypercube. This motivates the development of the serial test for uniformity in higher dimensions. To perform the serial test,

1. Divide  $(0,1)$  into  $k$  subintervals of equal size.
2. Generate  $n$  vectors of pseudorandom numbers each of size  $d$ ,

$$\begin{aligned}\vec{U}_1 &= (U_1, U_2, \dots, U_d) \\ \vec{U}_2 &= (U_{d+1}, U_{d+2}, \dots, U_{2d}) \\ &\vdots \\ \vec{U}_n &= (U_{(n-1)d+1}, U_{(n-1)d+2}, \dots, U_{nd})\end{aligned}$$

3. Let  $c_{j_1, j_2, \dots, j_d}$  be the count of the number of  $\vec{U}_i$ 's having the first component in subinterval  $j_1$ , second component in subinterval  $j_2$ , etc.
4. Compute

$$\chi^2_0(d) = \frac{k^d}{n} \sum_{j_1=1}^k \sum_{j_2=1}^k \cdots \sum_{j_d=1}^k \left( c_{j_1, j_2, \dots, j_d} - \frac{n}{k^d} \right)^2 \quad (2.11)$$

5. Reject the hypothesis that the  $\vec{U}_i$ 's are uniformly distributed in the  $d$ -dimensional unit hypercube if  $\chi^2_0(d) > \chi^2_{\alpha, k^d - 1}$  or if the  $p$ -value  $P\{\chi^2_{k^d - 1} > \chi^2_0(d)\}$  is less than  $\alpha$ .

Law (2007) provided an algorithm for computing  $c_{j_1, j_2, \dots, j_d}$ . This test examines how uniformly the random numbers fill-up the multidimensional space. This is a very important property when applying simulation to the evaluation of multidimensional integrals as is often found in the physical sciences.

## EXAMPLE 2.5 Two-Dimensional Chi-Squared Test in R

Using the same data as in the previous example, perform a two-dimensional chi-squared test using the statistical package R. Use four intervals for each of the dimensions.

The code given in Listing 2.1 is liberally commented for understanding the commands and essentially utilizes some of the classification and tabulation functionality in R to compute Equation (2.11). The results in Listing 2.2 are for demonstration purposes only since the expected number in the intervals is less than 5. However, you can see from the interval tabulation that the counts for the 2D intervals are close to the expected. The  $p$ -value suggests that we would not reject the hypothesis that there is nonuniformity in the pairs of the PRNs. The R code can be generalized for a larger sample or for performing a higher dimensional test. The reader is encouraged to try to run the code for a larger data set.

**Listing 2.1 R Code for Example 2.5**

```

nd = 100 #number of data points
data <- read.table("u01data.txt") # read in the data
d = 2 # dimensions to test
n = nd/d # number of vectors
m = t(matrix(data$V1,nrow=d)) # convert to matrix and transpose
b = seq(0,1, by = 0.25) # setup the cut points
xg = cut(m[,1],b,right=FALSE) # classify the x dimension
yg = cut(m[,2],b,right=FALSE) # classify the y dimension
xy = table(xg,yg) # tabulate the classifications
k = length(b) - 1 # the number of intervals
en = n/(k^d) # the expected number in an interval
vxy = c(xy) # convert matrix to vector for easier summing
vxymen = vxy-en # subtract expected number from each element
vxymen2 = vxymen*vxymen # square each element
schi = sum(vxymen2) # compute sum of squares
chi = schi/en # compute the chi-square test statistic
dof = (k^d) - 1 # compute the degrees of freedom
pv = pchisq(chi,dof, lower.tail=FALSE) # compute the p-value
# print out the results
cat("#observations = ", nd, "\n")
cat("#vectors = ", n, "\n")
cat("size of vectors, d = ", d, "\n")
cat("#intervals = ", k, "\n")
cat("cut points = ", b, "\n")
cat("expected # in each interval, n/k^d = ", en, "\n")
cat("interval tabulation = \n")
print(xy)
cat("\n")
cat("chisq value =", chi, "\n")
cat("dof =", dof, "\n")
cat("p-value = ", pv, "\n")

```

**2.4.1.3 Kolmogorov–Smirnov Test** The K-S test compares the hypothesized distribution,  $\hat{F}(x)$ , to the empirical distribution and does not depend on specifying intervals for tabulating the test statistic. The K-S test compares the theoretical CDF to the empirical CDF by checking the largest absolute deviation between the two over the range of the random variable. The K-S test is described in detail in Law (2007), which also includes a discussion of the advantages/disadvantages of the test. For example, Law (2007) indicated that the K-S test is more powerful than the chi-squared test and has the ability to be used on smaller sample sizes.

To apply the K-S test, we must be able to compute the empirical distribution function. The empirical distribution is the proportion of the observations that are less than or equal to  $x$ . Recalling Equation (2.7), we can define the empirical distribution as in Equation (2.12).

$$\tilde{F}_n(x) = \frac{c(\vec{x} \leq x)}{n} \quad (2.12)$$

**Listing 2.2 Results from R Code for Example 2.5**

```

Output:
#observations = 100
#vectors = 50
size of vectors, d = 2
#intervals = 4
cut points = 0 0.25 0.5 0.75 1
expected # in each interval, n/k^d = 3.125
interval tabulation =
  Yg
xg      [0,0.25) [0.25,0.5) [0.5,0.75) [0.75,1)
[0,0.25)      5        0        3        2
[0.25,0.5)    2        1        2        7
[0.5,0.75)    3        3        3        7
[0.75,1)      4        1        4        3

chisq value = 18.48
dof = 15
p-value = 0.2382715

```

To formalize this definition, suppose we have a sample of data  $x_i$  for  $i = 1, 2, \dots, n$  and then sort this data to obtain  $x_{(i)}$  for  $i = 1, 2, \dots, n$ , where  $x_{(1)}$  is the smallest,  $x_{(2)}$  is the second smallest, and so forth. Thus,  $x_{(n)}$  will be the largest value. These sorted numbers are called the *order statistics* for the sample and  $x_{(i)}$  is the  $i$ th order statistic.

Since the empirical distribution function is characterized by the proportion of the data values that are less than or equal to the  $i$ th order statistic for each  $i = 1, 2, \dots, n$ , Equation (2.12) can be rewritten as

$$\tilde{F}_n(x_{(i)}) = \frac{i}{n} \quad (2.13)$$

The reason that this revised definition works is because for a given  $x_{(i)}$ , the number of data values less than or equal to  $x_{(i)}$  will be  $i$  by definition of the order statistic. For each order statistic, the empirical distribution can be easily computed as follows:

$$\begin{aligned}\tilde{F}_n(x_{(1)}) &= \frac{1}{n} \\ \tilde{F}_n(x_{(2)}) &= \frac{2}{n} \\ &\vdots \\ \tilde{F}_n(x_{(i)}) &= \frac{i}{n} \\ &\vdots \\ \tilde{F}_n(x_{(n)}) &= \frac{n}{n} = 1\end{aligned}$$

A continuity correction is often used when defining the empirical distribution as follows:

$$\tilde{F}_n(x_{(i)}) = \frac{i - 0.5}{n}$$

This enhances the testing of continuous distributions. The sorting and computing of the empirical distribution is easy to accomplish in a spreadsheet program or the statistical package R.

The K-S test statistic,  $D_n$ , is defined as  $D_n = \max\{D_n^+, D_n^-\}$  where

$$\begin{aligned} D_n^+ &= \max_{1 \leq i \leq n} \{\tilde{F}_n(x_{(i)}) - \hat{F}(x_{(i)})\} \\ &= \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - \hat{F}(x_{(i)}) \right\} \\ D_n^- &= \max_{1 \leq i \leq n} \{\hat{F}(x_{(i)}) - \tilde{F}_n(x_{(i-1)})\} \\ &= \max_{1 \leq i \leq n} \left\{ \hat{F}(x_{(i)}) - \frac{i-1}{n} \right\} \end{aligned}$$

The K-S test statistic,  $D_n$ , represents the largest vertical distance between the hypothesized distribution and the empirical distribution over the range of the distribution. Appendix B contains critical values for the K-S test, where you reject the null hypothesis if  $D_n$  is greater than the critical value  $D_\alpha$ , where  $\alpha$  is the Type I significance level.

The K-S test statistic is relatively easy to compute using a spreadsheet, provided the CDF of the hypothesized distribution is available. Intuitively, a large value for the K-S test statistic indicates a poor fit between the empirical and the hypothesized distributions. The null hypothesis is that the data comes from the hypothesized distribution. While the K-S test can also be applied to discrete data, special tables must be used for getting the critical values. Additionally, the K-S test in its original form assumes that the parameters of the hypothesized distribution are known, that is, given without estimating from the data. Research on the effect of using the K-S test with estimated parameters has indicated that it will be conservative in the sense that the actual Type I error will be less than the specified one.

When applying the K-S test to test pseudorandom numbers, the hypothesized distribution is the uniform distribution on 0 to 1. The CDF for a uniform distribution on the interval  $(a, b)$  is given by

$$F(x) = P\{X \leq x\} = \frac{x-a}{b-a} \quad \text{for } a < x < b \quad (2.14)$$

Thus, for  $a = 0$  and  $b = 1$ , we have that  $F(x) = x$  for the  $U(0, 1)$  distribution. This simplifies the calculation of  $D_n^+$  and  $D_n^-$  to the following:

$$\begin{aligned} D_n^+ &= \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - \hat{F}(x_{(i)}) \right\} \\ &= \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - x_{(i)} \right\} \\ D_n^- &= \max_{1 \leq i \leq n} \left\{ \hat{F}(x_{(i)}) - \frac{i-1}{n} \right\} \\ &= \max_{1 \leq i \leq n} \left\{ x_{(i)} - \frac{i-1}{n} \right\} \end{aligned}$$

 **EXAMPLE 2.6 K-S Test for Hypothesis that PRNs are  $U(0, 1)$**

Given the data from Example 2.4, test the hypothesis that the data appears  $U(0, 1)$  versus that it is not  $U(0, 1)$  using the K-S goodness-of-fit test at the  $\alpha = 0.05$  significance level.

A good way to organize the computations is in a tabular form, which also facilitates the use of a spreadsheet. The second column is constructed by sorting the data. Recall that because we are testing the  $U(0, 1)$  distribution,  $F(x) = x$  and thus the fifth column is simply  $F(x_{(i)}) = x_{(i)}$ . The rest of the columns follow accordingly.

$i$	$x_{(i)}$	$i/n$	$\frac{i-1}{n}$	$F(x_{(i)})$	$\frac{i}{n} - F(x_{(i)})$	$F(x_{(i)}) - \frac{i-1}{n}$
1	0.006	0.010	0.000	0.006	0.004	0.006
2	0.015	0.020	0.010	0.015	0.005	0.005
3	0.016	0.030	0.020	0.016	0.014	-0.004
4	0.017	0.040	0.030	0.017	0.023	-0.013
5	0.034	0.050	0.040	0.034	0.016	-0.006
:	:	:	:	:	:	:
95	0.933	0.950	0.940	0.933	0.017	-0.007
96	0.938	0.960	0.950	0.938	0.022	-0.012
97	0.940	0.970	0.960	0.940	0.030	-0.020
98	0.951	0.980	0.970	0.951	0.029	-0.019
99	0.952	0.990	0.980	0.952	0.038	-0.028
100	0.971	1.000	0.990	0.971	0.029	-0.019

Computing  $D_n^+$  and  $D_n^-$  yields

$$D_n^+ = \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - x_{(i)} \right\} \\ = 0.108$$

$$D_n^- = \max_{1 \leq i \leq n} \left\{ x_{(i)} - \frac{i-1}{n} \right\} \\ = 0.038$$

Thus, we have that

$$D_n = \max\{D_n^+, D_n^-\} = \max\{0.108, 0.038\} = 0.108$$

Referring to Appendix B and using the approximation for sample sizes greater than 35, we have that  $D_{0.05} \approx 1.35/\sqrt{n}$ . Thus,  $D_{0.05} \approx 1.35/\sqrt{100} = 0.135$ . Since  $D_n < D_{0.05}$ , we would not reject the hypothesis that the data is uniformly distributed over the range from 0 to 1.

The K-S test performed in the solution to Example 2.6 can also be readily performed using the statistical software *R*. Assuming that the file *u01data.txt* contains the data for this example, then the following R commands will perform the test:

```

> data <- read.table("u01data.txt")
> ks.test(data, "punif", 0, 1)

One-sample Kolmogorov-Smirnov test

data: data
D = 0.1081, p-value = 0.1932
alternative hypothesis: two-sided

```

Since the  $p$ -value for the test is greater than  $\alpha = 0.05$ , we would not reject the hypothesis that the data is  $U(0, 1)$ .

## 2.4.2 Testing Independence

Pseudorandom numbers also need to be independently distributed from  $U(0, 1)$ . A variety of methods have been devised to examine the independence properties of random variables and pseudorandom numbers, in particular. The approaches can be classified into two major methods: (i) testing for independence in the time series and (ii) testing for patterns in the data. We will examine independence via an autocorrelation plot and then briefly discuss other methods.

**2.4.2.1 Autocorrelation Plot** An autocorrelation plot allows the dependence within the data to be quickly examined. An autocorrelation plot is a time series assessment tool that plots the lag- $k$  correlation versus the lag number. In order to understand these terms, we need to provide some background on how to think about a time series. A time series is a sequence of observations ordered by observation number,  $X_1, X_2, \dots, X_n$ . A time series  $X_1, X_2, \dots, X_n$  is said to be *covariance stationary* if

- The mean of  $X_i$ ,  $E[X_i]$ , exists and is a constant with respect to time. That is,  $\mu = E[X_i]$  for  $i = 1, 2, \dots$
- The variance of  $X_i$ ,  $Var[X_i]$ , exists and is constant with respect to time. That is,  $\sigma^2 = Var[X_i]$  for  $i = 1, 2, \dots$
- The lag- $k$  autocorrelation,  $\rho_k = Corr[X_i X_{i+k}]$ , is not a function of time  $i$  but is a function of the distance  $k$  between points. That is, the correlation between any two points in the series does not depend on where the points are in the series, it depends only on the distance between them in the series.

Recall that the correlation between two random variables is defined as

$$Corr[XY] = \frac{Cov[XY]}{\sqrt{Var[X]Var[Y]}} \quad (2.15)$$

where the covariance,  $Cov[XY]$ , is defined by

$$Cov[XY] = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y] \quad (2.16)$$

The correlation between two random variables  $X$  and  $Y$  measures the strength of linear association. The correlation has no units of measure and has a range:  $[-1, 1]$ . If the correlation between the random variables is less than zero, then the random variables are said to be

*negatively correlated.* This implies that if  $X$  tends to be high, then  $Y$  will tend to be low or, alternatively, if  $X$  tends to be low, then  $Y$  will tend to be high. If the correlation between the random variables is positive, the random variables are said to be *positively correlated*. This implies that if  $X$  tends to be high, then  $Y$  will tend to be high or, alternatively, if  $X$  tends to be low, then  $Y$  will tend to be low. If the correlation is zero, then the random variables are said to be uncorrelated. If  $X$  and  $Y$  are independent random variables, then the correlation between them will be zero. The converse of this is not necessarily true, but an assessment of the correlation should tell you something about the linear dependence between the random variables.

The autocorrelation between two random variables that are  $k$  time points apart in a covariance stationary time series is given by

$$\begin{aligned}\rho_k &= \text{Corr}[X_i X_{i+k}] = \frac{\text{Cov}[X_i X_{i+k}]}{\sqrt{\text{Var}[X_i] \text{Var}[X_{i+k}]}} \\ &= \frac{\text{Cov}[X_i X_{i+k}]}{\sigma^2} \text{ for } k = 1, 2, \dots\end{aligned}\quad (2.17)$$

A plot of  $\rho_k$  for increasing values of  $k$  is called an autocorrelation plot. The autocorrelation function as defined above is the theoretical function. When you have data, you must estimate the values of  $\rho_k$  from the actual times series. This involves forming an estimator for  $\rho_k$ . Law [2007] suggested plotting

$$\hat{\rho}_k = \frac{\hat{C}_k}{S^2(n)} \quad (2.18)$$

where

$$\hat{C}_k = \frac{1}{n-k} \sum_{i=1}^{n-k} (X_i - \bar{X}(n))(X_{i+k} - \bar{X}(n)) \quad (2.19)$$

$$S^2(n) = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X}(n))^2 \quad (2.20)$$

$$\bar{X}(n) = \frac{1}{n} \sum_{i=1}^n X_i \quad (2.21)$$

are the sample covariance, sample variance, and sample average, respectively.

Some time series analysis books, see, for example, Box et al. [1994], have a slightly different definition of the sample autocorrelation function:

$$r_k = \frac{c_k}{c_0} = \frac{\sum_{i=1}^{n-k} (X_i - \bar{X}(n))(X_{i+k} - \bar{X}(n))}{\sum_{i=1}^n (X_i - \bar{X}(n))^2} \quad (2.22)$$

where

$$c_k = \frac{1}{n} \sum_{i=1}^{n-k} (X_i - \bar{X}(n))(X_{i+k} - \bar{X}(n)) \quad (2.23)$$

Notice that the numerator in Equation (2.22) has  $n - k$  terms and the denominator has  $n$  terms. A plot of  $r_k$  versus  $k$  is called a correlogram or sample autocorrelation plot. For the data to be uncorrelated,  $r_k$  should be approximately zero for the values of  $k$ . Unfortunately, estimators of  $r_k$  are often highly variable, especially for large  $k$ ; however, an autocorrelation plot should still provide you with a good idea of independence.

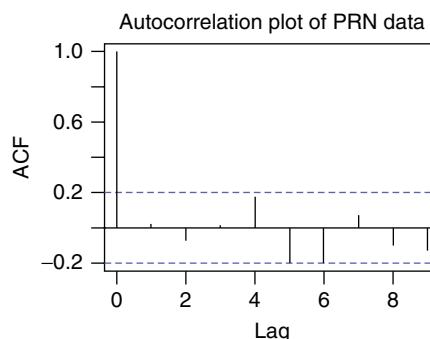
Formal tests can be performed using various time series techniques and their assumptions. A simple test can be performed based on the assumption that the series is white noise,  $N(0, 1)$  with all  $\rho_k = 0$ . Box et al. [1994] indicated that for large  $n$ ,  $Var(r_k) \approx \frac{1}{n}$ . Thus, a quick test of dependence is to check if sampled correlations fall within a reasonable confidence band around zero. For example, suppose  $n = 100$ , then  $Var(r_k) \approx \frac{1}{100} = 0.01$ . Then, the standard deviation is  $\sqrt{0.01} = 0.1$ . Assuming an approximate, 95% confidence level, yields a confidence band of  $\pm 1.645 \times 0.1 = \pm 0.1645$  about zero. Therefore, as long as the plotted values for  $r_k$  do not fall outside of this confidence band, it is likely that the data are independent.

A sample autocorrelation plot can be easily developed once the autocorrelations have been computed. Generally, the maximum lag should be set to no larger than one-tenth of the size of the data set because the estimation of higher lags is generally unreliable.

An autocorrelation plot can be easily performed using the statistical software R. Assuming that the file *u01data.txt* contains the data for this example, then the following R commands will make an autocorrelation plot. The resulting plot is shown in Figure 2.2. The  $r_0$  value represents the estimated variance. As can be seen in the figure, the `acf()` function of R automatically places the confidence band within the plot. In this instance, since none of the  $r_k, k \geq 1$  are outside the confidence band, we can conclude that the data are likely independent observations.

```
> data <- read.table("u01data.txt")
> rho = acf(data, main = "Autocorrelation Plot of PRN Data",
lag.max = 9)
> rho
Autocorrelations of series data, by lag
      0      1      2      3      4      5      6      7      8      9
1.000 0.015 -0.068 0.008 0.179 -0.197 -0.187 0.066 -0.095 -0.120
```

**2.4.2.2 Pattern Testing** A full exposition of testing for patterns in pseudorandom numbers is beyond the scope of this text. An overview is provided here with some references.



**Figure 2.2** Autocorrelation plot for *u01data.txt* data set.

**TABLE 2.1 Example Runs Up and Runs Down**

0.90	0.13	0.27	0.41	0.71	0.28	0.18	0.22	0.26	0.19	0.61	0.87	0.95	0.21	0.79
–	+	+	+	–	–	–	+	–	+	+	+	+	–	+

The autocorrelation test examines serial dependence; however, sequences that do not have other kinds of patterns are also desired. For example, the runs test attempts to test “upward” or “downward” patterns. The runs up and down test count the number of runs up, number of runs down, or sometimes just total number of runs versus expected number of runs. A run is a succession of similar events preceded and followed by a different event. The length of a run is the number of events in the run. Table 2.1 illustrates how to compute runs up and runs down for a simple sequence of numbers. In the table, a sequence of “–” indicates a run down and a sequence of “+” indicates a run up. In the table, there are eight runs (four runs up and four runs down).

Digit patterns can also be examined. The gap test counts the number of digits that appear between repetitions of a particular digit and uses a K-S test statistic to compare with the expected number of gaps. The poker test examines for independence based on the frequency with which certain digits are repeated in a series of numbers (e.g., pairs, and three of a kind). Banks et al. [2005] discussed how to perform these tests.

Does all this testing really matter? Yes! You should always know what generator you are relying on for your simulation analysis. The development and testing of random number generators is serious business. You should stick with well-researched generators and reliable well-tested software.

## 2.5 GENERATING RANDOM VARIATES FROM DISTRIBUTIONS

In simulation, pseudorandom numbers serve as the foundation for generating samples from probability distribution models. We will now assume that the random number generator has been rigorously tested and that it produces sequences of  $U_i \sim U(0, 1)$  numbers. We now want to take the  $U_i \sim U(0, 1)$  and utilize them to generate from probability distributions.

The realized value from a probability distribution is called a random variate. Simulations use many different probability distributions as inputs. Thus, methods for generating random variates from distributions are required, that is, how do you generate samples from probability distributions? In generating random variates, the goal is to produce samples  $X_i$  from a distribution  $F(x)$ , given a source of random numbers,  $U_i \sim U(0, 1)$ . There are four basic strategies or methods for producing random variates:

1. Inverse transform or inverse CDF method
2. Convolution
3. Acceptance/rejection
4. Mixture and truncated distributions.

The following sections discuss each of these methods.

### 2.5.1 Inverse Transform

The inverse transform method is the preferred method for generating random variates, provided that the inverse transform of the CDF can be easily derived or computed numerically. The key advantage for the inverse transform method is that for every  $U_i$  generated, a corresponding  $X_i$  will be produced.

The inverse transform technique utilizes the inverse of the CDF as illustrated in Figure 2.3. First, generate a number  $u_i$  between 0 and 1 (along the  $U$ -axis) and then find the corresponding  $x_i$  coordinate by using  $F^{-1}(\cdot)$ . For various values of  $u_i$ , the  $x_i$  will be properly “distributed” along the  $x$ -axis. The beauty of this method is that there is a one-to-one mapping between  $u_i$  and  $x_i$ . In other words, for each  $u_i$ , there is a unique  $x_i$  because of the monotone property of the CDF.

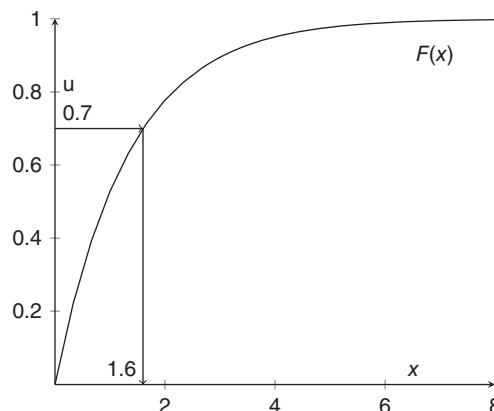
The idea illustrated in Figure 2.3 is based on the following theorem.

**Theorem 2.2 (Inverse Transform Theorem)** *Let  $X$  be a random variable with  $X \sim F(x)$ . Define another random variable  $Y$  such that  $Y = F(X)$ . That is,  $Y$  is determined by evaluating the function  $F(\cdot)$  at the value  $X$ . If  $Y$  is defined in the manner, then  $Y \sim U(0, 1)$ .*

*Proof. (Proof of Theorem 2.2):* The proof utilizes the definition of the CDF to derive the CDF for  $Y$ .

$$\begin{aligned}
 F(y) &= P\{Y \leq y\} \\
 &= P\{F(X) \leq y\} \text{ substitute for } Y \\
 &= P\{F^{-1}(F(X)) \leq F^{-1}(y)\} \text{ apply inverse} \\
 &= P\{X \leq F^{-1}(y)\} \text{ definition of inverse} \\
 &= F(F^{-1}(y)) \text{ definition of CDF} \\
 &= y \text{ definition of inverse}
 \end{aligned}$$

Since  $P(Y \leq y) = y$  defines a  $U(0, 1)$  random variable, the proof is complete. ■



**Figure 2.3** Illustration of inverse transform method.

This result also works in reverse; if you start with a uniformly distributed random variable, then you can get a random variable with the distribution of  $F(x)$ . The idea is to generate  $U_i \sim U(0, 1)$  and then to use the inverse CDF to transform the random number to the appropriately distributed random variate.

---

**Exhibit 2.1** Continuous Inverse CDF Method
 

---

```

1 : u = rand()
2 : x = F-1(u)
3 : RETURN x
  
```

---

Let us assume that we have a function, *rand()*, that will provide pseudorandom numbers. Then, Exhibit 2.1 presents the pseudo-code for the inverse transform algorithm. Line 1 generates a uniform number. Line 2 takes the inverse, and line 3 returns the random variable.

Example 2.7 illustrates the inverse transform method for the exponential distribution. Exhibit 2.2 illustrates the inverse transform algorithm for the exponential distribution.

### EXAMPLE 2.7 Inverse CDF Method for Exponential Distribution

The exponential distribution is often used to model the time until an event (e.g., time until failure and time until an arrival) and has the form:

$$f(x) = \begin{cases} 0.0 & \text{if } x < 0 \\ \lambda e^{-\lambda x} & \text{if } x \geq 0 \end{cases}$$

with

$$\text{E}[X] = \frac{1}{\lambda} \quad \text{Var}[X] = \frac{1}{\lambda^2}$$

Derive the inverse CDF and present an inverse CDF algorithm for the exponential distribution. Suppose that  $\lambda = 0.75$  and  $u = 0.7$  represented a pseudorandom number, generate an exponentially distributed random variate for this situation.

To apply the inverse CDF method, you must first compute the CDF.

For  $x < 0$ ,

$$F(x) = P\{X \leq x\} = \int_{-\infty}^x f(u) du = \int_{-\infty}^x 0 du = 0$$

For  $x \geq 0$ ,

$$\begin{aligned} F(x) &= P\{X \leq x\} = \int_{-\infty}^x f(u) du \\ &= \int_{-\infty}^0 f(u) du + \int_0^x f(u) du \\ &= \int_0^x \lambda e^{\lambda u} du = - \int_0^x e^{-\lambda u} (-\lambda) du \\ &= -e^{\lambda u} \Big|_0^x = -e^{\lambda x} - (-e^0) = 1 - e^{-\lambda x} \end{aligned}$$

Thus, the CDF of the exponential distribution is

$$F(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 - e^{-\lambda x} & \text{if } x \geq 0 \end{cases}$$

Now the inverse of the CDF can be derived by setting  $u = F(x)$  and solving for  $x = F^{-1}(u)$ .

$$\begin{aligned} u &= 1 - e^{-\lambda x} \\ x &= \frac{-1}{\lambda} \ln(1 - u) = F^{-1}(u) \end{aligned}$$

Suppose that  $\lambda = 0.75$  and we have  $u = 0.7$ , then the generated  $x$  would be

$$x = \frac{-1}{0.75} \ln(1 - 0.7) = 1.6053$$

---

**Exhibit 2.2** *Expo( $1/\lambda$ ) Inverse CDF Method*


---

- 1:  $u = U(0, 1)$
  - 2:  $x = \frac{-1}{\lambda} \ln(1 - u)$
  - 3: RETURN  $x$
- 

### EXAMPLE 2.8 Inverse Transform Method for $U(a, b)$

The uniform distribution over an interval  $(a, b)$  is often used to model situations where the analyst does not have much information and can only assume that the outcome is equally likely over a range of values. The uniform distribution has the following characteristics:

$$X \sim \text{Uniform}(a, b)$$

$$f(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

$$\text{E}[X] = \frac{a+b}{2} \quad \text{Var}[X] = \frac{(b-a)^2}{12}$$

$$F(x) = \begin{cases} 0.0 & x < a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ 1.0 & x > b \end{cases}$$

Derive the inverse CDF and present an inverse CDF algorithm for the  $U(a, b)$  distribution. Suppose that  $a = 5$ ,  $b = 35$ , and  $u = 0.25$  represented a pseudorandom number, generate a  $U(5, 35)$  distributed random variate for this situation.

The inverse of the CDF can be derived by setting  $u = F(x)$  and solving for  $x = F^{-1}(u)$ .

$$u = \frac{x - a}{b - a}$$

$$u(b - a) = x - a$$

$$x = a + u(b - a) = F^{-1}(u)$$

Suppose that  $a = 5$  and  $b = 35$ . Suppose that  $u = 0.25$ , then the generated  $x$  would be

$$F^{-1}(u) = x = 5 + 0.25 \times (35 - 5) = 5 + 7.5 = 12.5$$

Notice how the value of  $u$  is first scaled on the range  $(0, b - a)$  and then shifted to the range  $(a, b)$ . For the uniform distribution, this transformation is linear because of the form of its  $F(x)$ . Exhibit 2.3 summarizes the inverse transform algorithm for the continuous  $U(a, b)$  distribution.

---

**Exhibit 2.3  $U(a, b)$  Inverse CDF Method**


---

- 1 :  $u = U(0, 1)$
  - 2 :  $x = a + u(b - a)$
  - 3 : RETURN  $x$
- 

The inverse CDF method also works for discrete distributions. For a discrete random variable,  $X$ , with possible values  $x_1, x_2, \dots, x_n$  ( $n$  may be infinite), the probability distribution is called the probability mass function (PMF) and denoted

$$f(x_i) = P(X = x_i)$$

where  $f(x_i) \geq 0$  for all  $x_i$  and

$$\sum_{i=1}^n f(x_i) = 1$$

The CDF is

$$F(x) = P(X \leq x) = \sum_{x_i \leq x} f(x_i)$$

and satisfies  $0 \leq F(x) \leq 1$ , and if  $x \leq y$ , then  $F(x) \leq F(y)$ . In order to apply the inverse transform method to discrete distributions, the CDF can be searched to find the value of  $x$  associated with the given  $u$ . This process is illustrated in Example 2.9.

### EXAMPLE 2.9 Inverse CDF Method for Discrete Distribution

Suppose you have a random variable,  $X$ , with the following discrete PMF and CDF.

$x_i$	1	2	3	4
$f(x_i)$	0.4	0.3	0.2	0.1

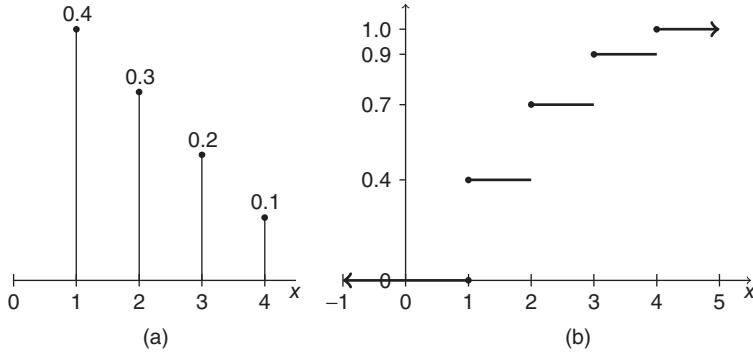


Figure 2.4 Example of (a) PMF and (b) CDF.

Plot the PMF and CDF for this random variable. Then develop an inverse CDF for generating random numbers from this distribution. Finally, given  $u_1 = 0.934$  and  $u_2 = 0.1582$  are pseudorandom numbers, generate the two corresponding random variates.

The functional form of the PMF and CDF are given as follows:

$$P\{X = x\} = \begin{cases} 0.4 & x = 1 \\ 0.3 & x = 2 \\ 0.2 & x = 3 \\ 0.1 & x = 4 \end{cases} \quad F(x) = \begin{cases} 0.0 & \text{if } x < 1 \\ 0.4 & \text{if } 1 \leq x < 2 \\ 0.7 & \text{if } 2 \leq x < 3 \\ 0.9 & \text{if } 3 \leq x < 4 \\ 1.0 & \text{if } x \geq 4 \end{cases}$$

Figure 2.4 illustrates the PMF and the CDF for this discrete distribution. Examining Figure 2.4 indicates that for any value of  $u_i$  in the interval  $(0.4, 0.7]$ , you get an  $x_i$  of 2. Thus, generating random numbers from this distribution can be accomplished by using Equation (2.24) which represents the inverse of the CDF.

$$F^{-1}(u) = \begin{cases} 1 & \text{if } 0.0 \leq u \leq 0.4 \\ 2 & \text{if } 0.4 < u \leq 0.7 \\ 3 & \text{if } 0.7 < u \leq 0.9 \\ 4 & \text{if } 0.9 < u \leq 1.0 \end{cases} \quad (2.24)$$

Suppose  $u_1 = 0.934$ , then by Equation (2.24),  $x = 4$ . If  $u_2 = 0.1582$ , then  $x = 1$ . Thus, we use Equation (2.24) to look up the appropriate  $x$  for a given  $u$ .

For a discrete distribution, given a value for  $u$ , pick  $x_i$ , such that  $F(x_{i-1}) < u \leq F(x_i)$  provides the inverse CDF function. Thus, for any given value of  $u$ , the generation process amounts to a lookup table of the corresponding value for  $x_i$ . This simply involves searching until the condition  $F(x_{i-1}) < u \leq F(x_i)$  is true. Since  $F(x)$  is an increasing function in  $x$ , only the upper limit needs to be checked. Exhibit 2.4 presents these ideas in the form of an algorithm.

In Exhibit 2.4, if the test  $F(x) \leq u$  is true, the while loop moves to the next interval. If the test failed,  $u > F(x)$  must be true. The while loop stops and  $x$  is the last value checked,

---

**Exhibit 2.4** Discrete Inverse CDF

---

```

1: u = rand()
2: i = 1
3: x = xi
4: WHILE F(x) ≤ u
5:     i = i + 1
6:     x = xi
7: END WHILE
8: RETURN x

```

---

which is returned. Thus, only the upper limit in the next interval needs to be tested. Other more complicated and possibly more efficient methods for performing this process are discussed in Fishman [2006] and Ripley [1987].

Using the inverse transform method, for discrete random variables, a Bernoulli random variate can be easily generated as shown in Exhibit 2.5.

---

**Exhibit 2.5** Bernoulli( $p$ ) Inverse CDF Method

---

```

1: u = rand()
2: IF (u ≤ p) THEN
3:     x = 1
4: ELSE
5:     x = 0
6: END IF
7: RETURN x

```

---

To generate discrete uniform random variables, the inverse transform method yields the algorithm provided in Exhibit 2.6.

---

**Exhibit 2.6** Discrete  $U(a, b)$  Inverse CDF Method

---

```

1: u = rand()
2: x = a + ⌊(b - a + 1)u⌋
3: RETURN x

```

---

The inverse transform method also works for generating geometric random variables. Unfortunately, the geometric distribution has multiple definitions. Let  $X$  be the number of Bernoulli trials needed to get one success. Thus,  $X$  has range 1, 2, ... and distribution

$$P\{X = k\} = (1 - p)^{k-1} p \quad (2.25)$$

We will call this the shifted geometric in this text. The algorithm to generate a shifted geometric random variables is presented in Exhibit 2.7.

If the geometric is defined as the number of failures  $Y = X - 1$  before the first success, then  $Y$  has range 0, 1, 2, ... and probability distribution:

$$P\{Y = k\} = (1 - p)^k p \quad (2.26)$$

We will call this distribution the geometric distribution in this text. The algorithm to generate a geometric random variables is presented in Exhibit 2.8.

---

**Exhibit 2.7** Shifted Geometric( $p$ ) Inverse CDF Method

---

```

1: u = rand()
2: x = 1 + ⌊ ln(1-u) ⌋
   ln(1-p)
3: RETURN x

```

---



---

**Exhibit 2.8** Geometric( $p$ ) Inverse CDF Method

---

```

1: u = rand()
2: y = ⌊ ln(1-u) ⌋
   ln(1-p)
3: RETURN y

```

---

The Poisson distribution is often used to model the number of occurrences within an interval time, space, etc. For example,

- the number of phone calls to a doctor's office in an hour,
- the number of persons arriving to a bank in a day,
- the number of cars arriving to an intersection in an hour,
- the number of defects that occur within a length of item,
- the number of typos in a book,
- the number of pot holes in a mile of road.

Assume that we have an interval of real numbers and that incidents occur at random throughout the interval. If the interval can be partitioned into subintervals of small enough length such that

- the probability of more than one incident in a subintervals is zero,
- the probability of one incident in a subintervals is the same for all intervals and proportional to the length of the subintervals, and
- the number of incidents in each subintervals is independent of other subintervals,

then, we have a Poisson distribution. Let the probability of an incident falling into a subinterval be  $p$ . Let there be  $n$  subintervals. An incident either falls in a subinterval or it does not. This can be considered a Bernoulli trial. Suppose there are  $n$  subintervals, then the number of incidents that fall in the large interval is a binomial random variable with expectation  $n * p$ . Let  $\lambda = np$  be a constant and keep dividing the main interval into smaller and smaller subintervals such that  $\lambda$  remains constant. To keep  $\lambda$  constant, increase  $n$  and decrease  $p$ . What is the chance that  $x$  incidents occur in the  $n$  subintervals?

$$\binom{n}{x} \left(\frac{\lambda}{n}\right)^x \left(1 - \frac{\lambda}{n}\right)^{n-x}$$

Take the limit as  $n$  goes to infinity

$$\lim_{n \rightarrow \infty} \binom{n}{x} \left(\frac{\lambda}{n}\right)^x \left(1 - \frac{\lambda}{n}\right)^{n-x}$$

and we get the Poisson distribution:

$$P\{X = x\} = \frac{e^{-\lambda} \lambda^x}{x!} \quad \lambda > 0, \quad x = 0, 1, \dots$$

$$\text{E}[X] = \lambda$$

$$\text{Var}[X] = \lambda$$

If a Poisson random variable represents the number of incidents in some interval, then the mean of the random variable must equal the expected number of incidents in the same length of interval. In other words, the units must match. When examining the number of incidents in a unit of time, the Poisson distribution is often written as

$$P\{X(t) = x\} = \frac{e^{-\lambda t} (\lambda t)^x}{x!}$$

where  $X(t)$  is the number of events that occur in  $t$  time units. This leads to an important relationship with the exponential distribution.

Let  $X(t)$  be a Poisson random variable that represents the number of arrivals in  $t$  time units with  $\text{E}[X(t)] = \lambda t$ . What is the probability of having no events in the interval from 0 to  $t$ ?

$$P\{X(t) = 0\} = \frac{e^{-\lambda t} (\lambda t)^0}{0!} = e^{-\lambda t}$$

This is the probability that no one arrives in the interval  $(0, t)$ . Let  $T$  represent the time until an arrival from any starting point in time. What is the probability that  $T > t$ ? That is, what is the probability that the time of the arrival is sometime after  $t$ ? For  $T$  to be bigger than  $t$ , we must not have anybody arrive before  $t$ . Thus, these two events are the same:  $\{T > t\} = \{X(t) = 0\}$ . Thus,  $P\{T > t\} = P\{X(t) = 0\} = e^{-\lambda t}$ . What is the probability that  $T \leq t$ ?

$$P\{T \leq t\} = 1 - P\{T > t\} = 1 - e^{-\lambda t}$$

This is the CDF of  $T$ , which is an exponential distribution. Thus, if  $T$  is a random variable that represents the time between events and  $T$  is exponential with mean  $1/\lambda$ , then the number of events in  $t$  will be a Poisson process with  $\text{E}[X(t)] = \lambda t$ . Therefore, a method for generating Poisson random variates with mean  $\lambda$  can be derived by counting the number of events that occur before  $t$  when the time between events is exponential with mean  $1/\lambda$ .

### EXAMPLE 2.10 Generating from a Poisson Distribution

Let  $X(t)$  represent the number of customers that arrive to a bank in an interval of length  $t$ , where  $t$  is measured in hours. Suppose  $X(t)$  has a Poisson distribution with mean rate  $\lambda = 4$  per hour. Use the first row of PRNs from Example 2.4 to generate a value of  $X(2)$ . That is, generate the number of arrivals in 2 hours.

Because of the relationship between the Poisson distribution and the exponential distribution, the time between events  $T$  will have an exponential distribution with mean  $0.25 = 1/\lambda$ .

Thus, we have

$$T_i = \frac{-1}{\lambda} \ln (1 - u_i) = -0.25 \ln (1 - u_i)$$

$$A_i = \sum_{k=1}^i T_k$$

where  $T_i$  represents the time between the  $i - 1$  and  $i$  arrivals and  $A_i$  represents the time of the  $i$ th arrival. Using the  $u_i$  from Example 2.4, we can compute  $T_i$  and  $A_i$  until  $A_i$  goes over 2 hours.

$i$	$u_i$	$T_i$	$A_i$
1	0.971	0.881	0.881
2	0.687	0.290	1.171
3	0.314	0.094	1.265
4	0.752	0.349	1.614
5	0.830	0.443	2.057

Since the arrival of the fifth customer occurs after 2 hours,  $X(2) = 4$ . That is, there were four customers who arrived within the 2 hours.

The inverse transform technique is general and is used when  $F^{-1}(\cdot)$  is in closed form and easy to compute. It also has the advantage of using one  $U(0, 1)$  for each  $X$  generated, which helps when applying certain techniques that are used to improve the estimation process in simulation experiments. Because of this advantage, many simulation languages utilize the inverse transform technique even if a closed-form solution to  $F^{-1}(\cdot)$  does not exist by numerically inverting the function.

### 2.5.2 Convolution

Many random variables are related to each other through some functional relationship. One of the most common relationships is the convolution relationship. The distribution of the sum of two or more random variables is called the *convolution*. Let  $Y_i \sim G(y)$  be IID random variables. Let  $X = \sum_{i=1}^n Y_i$ . Then the distribution of  $X$  is said to be the  $n$ -fold convolution of  $Y$ . Some common random variables that are related through the convolution operation are

- A binomial random variable is the sum of Bernoulli random variables.
- A negative binomial random variable is the sum of geometric random variables.
- An Erlang random variable is the sum of exponential random variables.
- A normal random variable is the sum of other normal random variables.
- A chi-squared random variable is the sum of squared normal random variables.

The basic convolution algorithm simply generates  $Y_i \sim G(y)$  and then sums the generated random variables. For example, suppose that  $Y_i \sim \exp(E[Y_i] = 1/\lambda)$ . That is,  $Y$  is exponentially distributed with rate parameter  $\lambda$ . Now, define  $X$  as  $X = \sum_{i=1}^r Y_i$ . One can show that  $X$  will have an Erlang distribution with parameters  $(r, \lambda)$ , where  $E[X] = r/\lambda$  and

$\text{Var}[X] = r/\lambda^2$ . Thus, an Erlang( $r, \lambda$ ) is an  $r$ -fold convolution of  $r$  exponentially distributed random variables with common mean  $1/\lambda$ .

### ■ EXAMPLE 2.11 Generating Erlang( $r, \lambda$ ) Random Variates

Using the following pseudorandom numbers  $u_1 = 0.35$ ,  $u_2 = 0.64$ , and  $u_3 = 0.14$ , generate a random variate from an Erlang distribution having parameters  $r = 3$  and  $\lambda = 0.5$ .

This requires generating three exponential distributed random variates each with  $\lambda = 0.5$  and adding them up.

$$\begin{aligned}y_1 &= \frac{-1}{\lambda} \ln (1 - u_1) = \frac{-1}{0.5} \ln (1 - 0.35) = 0.8616 \\y_2 &= \frac{-1}{\lambda} \ln (1 - u_2) = \frac{-1}{0.5} \ln (1 - 0.64) = 2.0433 \\y_3 &= \frac{-1}{\lambda} \ln (1 - u_3) = \frac{-1}{0.5} \ln (1 - 0.14) = 0.3016 \\x &= y_1 + y_2 + y_3 = 0.8616 + 2.0433 + 0.3016 = 3.2065\end{aligned}$$

A number of the exercises for this chapter invite the reader to generate random variates via the convolution method. Because of its simplicity, the convolution method is easy to implement; however, in a number of cases (in particular for large  $n$ ), there are more efficient algorithms available.

#### 2.5.3 Acceptance/Rejection

In the acceptance–rejection method, the probability density function (PDF),  $f(x)$ , from which it is desired to obtain a sample, is replaced by a proxy PDF,  $w(x)$ , which can be sampled from more easily. The following illustrates how  $w(x)$  is defined such that the *selected* samples from  $w(x)$  can be used directly to represent random variates from  $f(x)$ . The PDF  $w(x)$  is based on the development of a majorizing function for  $f(x)$ . A majorizing function,  $g(x)$ , for  $f(x)$ , is a function such that  $g(x) \geq f(x)$  for  $-\infty < x < +\infty$ . Figure 2.5 illustrates the concept of a majorizing function for  $f(x)$ , which simply means a function that is bigger than  $f(x)$  everywhere.

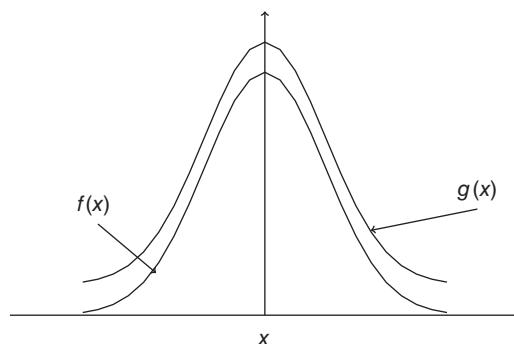


Figure 2.5 Illustration of a majorizing function.

In addition, to be a majorizing function for  $f(x)$ ,  $g(x)$  must have finite area. In other words,

$$c = \int_{-\infty}^{+\infty} g(x) dx \quad (2.27)$$

If  $w(x)$  is defined as  $w(x) = g(x)/c$ , then  $w(x)$  will be a PDF. The acceptance–rejection method starts by obtaining a random variate  $W$  from  $w(x)$ . Recall that  $w(x)$  should be chosen with the stipulation that it can be easily sampled, for example, via the inverse transform method. Let  $U \sim U(0, 1)$ . The steps of the procedure are as provided in Exhibit 2.9. The sampling of  $U$  and  $W$  continue until  $U \times g(W) \leq f(W)$  and  $W$  is returned. If  $U \times g(W) > f(W)$ , then the loop repeats.

---

**Exhibit 2.9** Acceptance–Rejection Algorithm

---

- 1: REPEAT
  - 2: Generate  $W \sim w(x)$
  - 3: Generate  $U \sim U(0, 1)$
  - 4: UNTIL  $U \times g(W) \leq f(W)$
  - 5: RETURN  $W$
- 

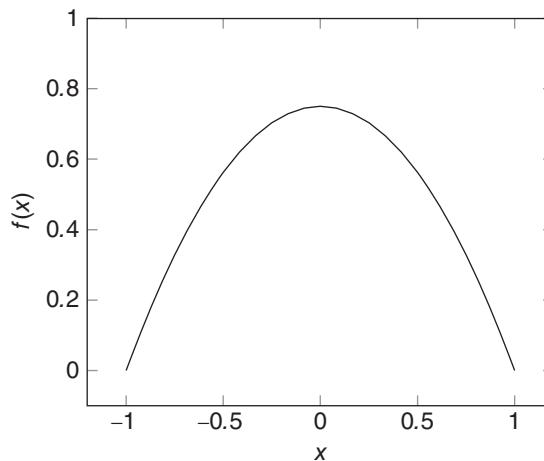
The validity of the procedure is based on deriving the CDF of  $W$ , given that the  $W = w$  was accepted,  $P\{W \leq x | W = w \text{ is accepted}\}$ . The proof is left as an exercise.

The efficiency of the acceptance–rejection method is enhanced as the probability of rejection is reduced. This probability depends directly on the choice of the majorizing function  $g(x)$ . The acceptance–rejection method has a nice intuitive geometric connotation, which is best illustrated with an example.

### EXAMPLE 2.12 Acceptance–Rejection Method

Consider the following PDF over the range  $[-1, 1]$ . Develop an acceptance/rejection-based algorithm for  $f(x)$ .

$$f(x) = \begin{cases} \frac{3}{4}(1 - x^2) & -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$



The first step in deriving an acceptance/rejection algorithm for  $f(x)$  is to select an appropriate majorizing function. A simple method to choose a majorizing function is to set  $g(x)$  equal to the maximum value of  $f(x)$ . As can be seen from the plot of  $f(x)$ , the maximum value of  $3/4$  occurs at  $x$  equal to 0. Thus, we can set  $g(x) = 3/4$ . In order to proceed, we need to construct the PDF associated with  $g(x)$ . Define  $w(x) = g(x)/c$  as the PDF. To determine  $c$ , we need to determine the area under the majorizing function

$$c = \int_{-1}^1 g(x)dx = \int_{-1}^1 \frac{3}{4}dx = \frac{3}{2}$$

Thus,

$$w(x) = \begin{cases} \frac{1}{2} & -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

This implies that  $w(x)$  is a uniform distribution over the range from  $[-1, 1]$ . Based on Example 2.8, a uniform distribution over the range from  $a$  to  $b$  can be generated with  $a + U(b - a)$ . Thus, for this case,  $a = -1$  and  $b = 1$ , with  $b - a = 1 - -1 = 2$ . The acceptance–rejection algorithm is given in Exhibit 2.10.

---

**Exhibit 2.10** A/R Algorithm for Example 2.12

---

```

1: REPEAT
2:   Generate  $U_1 \sim U(0, 1)$ 
3:    $W = -1 + 2U_1$ 
4:   Generate  $U_2 \sim U(0, 1)$ 
5:    $f = \frac{3}{4}(1 - W^2)$ 
6: UNTIL  $U_2 \times \frac{3}{4} \leq f$ 
7: RETURN  $\bar{W}$ 

```

---

Steps 2 and 3 of Exhibit 2.10 generate a random variate,  $W$ , from  $w(x)$ . Note that  $W$  is in the range  $[-1, 1]$  (the same as the range of  $X$ ) so that step 5 is simply finding the height associated with  $W$  in terms of  $f$ . Step 4 generates  $U_2$ . What is the range of  $U_2 \times \frac{3}{4}$ ? The range is  $[0, \frac{3}{4}]$ . Note that this range corresponds to the range of possible values for  $f$ . Thus, in step 6, a point between  $[0, \frac{3}{4}]$  is being compared to the candidate point's height  $f(W)$  along the vertical axis. If the point is under  $f(W)$ , the  $W$  is accepted; otherwise the  $W$  is rejected and another candidate point must be generated. In other words, if a point “under the curve” is generated, it will be accepted.

As illustrated in Example 2.12, the probability of acceptance is related to how close  $g(x)$  is to  $f(x)$ . The ratio of the area under  $f(x)$  to the area under  $g(x)$  is the probability of accepting. Since the area under  $f(x)$  is 1, the probability of acceptance,  $P_a$ , is

$$P_a = \frac{1}{\int_{-\infty}^{+\infty} g(x)dx} = \frac{1}{c} \quad (2.28)$$

where  $c$  is given by Equation (2.27). For Example 2.12, the probability of acceptance is  $P_a = 2/3$ . Based on this example, it should be clear that the more  $g(x)$  is similar to the PDF

$f(x)$ , the better (higher) the probability of acceptance. The key to efficiently generating random variates using the acceptance–rejection method is finding a suitable majorizing function over the same range as  $f(x)$ . In the example, this was easy because  $f(x)$  had a finite range. It is more challenging to derive an acceptance–rejection algorithm for a PDF,  $f(x)$ , if it has an infinite range because it may be more challenging to find a good majorizing function.

### 2.5.4 Mixture Distributions, Truncated Distributions, and Shifted Random Variables

This section describes three random variate generation methods that build on the previously discussed methods. These methods allow for more flexibility in modeling the underlying randomness. First, let us consider the definition of a mixture distribution and then consider some examples.

**Definition 2.4 (Mixture Distribution)** *The distribution of a random variable  $X$  is a mixture distribution if the CDF of  $X$  has the form*

$$F_X(x) = \sum_{i=1}^k \omega_i F_{X_i}(x) \quad (2.29)$$

where  $0 < \omega_i < 1$ ,  $\sum_{i=1}^k \omega_i = 1$ ,  $k \geq 2$ , and  $F_{X_i}(x)$  is the CDF of a continuous or discrete random variable  $X_i$ ,  $i = 1, \dots, k$ .

Notice that the  $\omega_i$  can be interpreted as a discrete probability distribution as follows. Let  $I$  be a random variable with range  $I \in \{1, \dots, k\}$  where  $P\{I = i\} = \omega_i$  is the probability that the  $i$ th distribution  $F_{X_i}(x)$  is selected. Then, the procedure for generating from  $F_X(x)$  is to randomly generate  $I$  from  $g(i) = P\{I = i\} = \omega_i$  and then generate  $X$  from  $F_{X_I}(x)$ . Exhibit 2.11 presents this procedure.

---

#### Exhibit 2.11 Mixture Distribution Algorithm

---

```

1: Generate  $I \sim g(i)$ 
2: Generate  $X \sim F_{X_I}(x)$ 
3: RETURN X

```

---

Because mixture distributions combine the characteristics of two or more distributions, they provide for more flexibility in modeling. For example, many of the standard distributions that are presented in introductory probability courses, such as the normal, Weibull, and lognormal, have a single mode. Mixture distributions are often utilized for the modeling of data sets that have more than one mode.

As an example of a mixture distribution, we will discuss the hyperexponential distribution. The hyperexponential is useful in modeling situations that have a high degree of variability. The coefficient of variation is defined as the ratio of the standard deviation to the expected value for a random variable  $X$ . The coefficient of variation is defined as  $c_v = \sigma/\mu$ , where  $\sigma = \sqrt{\text{Var}[X]}$  and  $\mu = E[X]$ . For the hyperexponential distribution,  $c_v > 1$ . The hyperexponential distribution is commonly used to model service times that have different (and mutually exclusive) phases. An example of this situation is paying with

a credit card or cash at a checkout register. Example 2.13 illustrates how to generate from a hyperexponential distribution.

### ■ EXAMPLE 2.13 Hyperexponential Distribution

Suppose the time that it takes to pay with a credit card,  $X_1$ , is exponentially distributed with a mean of 1.5 min and the time that it takes to pay with cash,  $X_2$ , is exponentially distributed with a mean of 1.1 min. In addition, suppose that the chance that a person pays with credit is 70%. Then, the overall distribution representing the payment service time,  $X$ , has an hyperexponential distribution with parameters  $\omega_1 = 0.7$ ,  $\omega_2 = 0.3$ ,  $\lambda_1 = 1/1.5$ , and  $\lambda_2 = 1/1.1$ .

$$\begin{aligned} F_X(x) &= \omega_1 F_{X_1}(x) + \omega_2 F_{X_2}(x) \\ F_{X_1}(x) &= 1 - \exp(-\lambda_1 x) \\ F_{X_2}(x) &= 1 - \exp(-\lambda_2 x) \end{aligned}$$

Derive an algorithm for this distribution. Assume that you have two pseudorandom numbers,  $u_1 = 0.54$  and  $u_2 = 0.12$ , generate a random variate from  $F_X(x)$ .

In order to generate a payment service time,  $X$ , we can use the algorithm presented in Exhibit 2.12.

---

#### Exhibit 2.12 Algorithm for Example 2.13

---

```

1: Generate  $u \sim U(0, 1)$ 
2: Generate  $v \sim U(0, 1)$ 
3: IF  $u \leq 0.7$  THEN
4:    $X = F_{X_1}^{-1}(v) = -1.5 \ln(1 - v)$ 
5: ELSE
6:    $X = F_{X_2}^{-1}(v) = -1.1 \ln(1 - v)$ 
7: END IF
8: RETURN  $X$ 
```

---

Using  $u_1 = 0.54$ , because  $0.54 \leq 0.7$ , we have that

$$X = F_{X_1}^{-1}(0.12) = -1.5 \ln(1 - 0.12) = 0.19175$$

In Example 2.13, generating  $X$  from  $F_{X_i}(x)$  utilizes the inverse transform method for generating from the two exponential distribution functions. In general,  $F_{X_i}(x)$  for a general mixture distribution might be any distribution. For example, we might have a mixture of a gamma and a lognormal distribution. To generate from the individual  $F_{X_i}(x)$ , one would use the most appropriate generation technique for that distribution. For example,  $F_{X_1}(x)$  might use inverse transform,  $F_{X_2}(x)$  might use acceptance/rejection, and  $F_{X_3}(x)$  might use convolution. This provides great flexibility in modeling and in generation.

In general, we may have situations where we need to control the domain over which the random variates are generated. For example, when we are modeling situations that involve time (as is often the case within simulation), we need to ensure that we do not generate negative values. Or, for example, it may be physically impossible to perform a task in a

time that is shorter than a particular value. The next two generation techniques assist with modeling these situations.

A *truncated distribution* is a distribution derived from another distribution for which the range of the random variable is restricted. Truncated distributions can be either discrete or continuous. The presentation here illustrates the continuous case. Suppose we have a random variable,  $X$  with PDF,  $f(x)$ , and CDF  $F(x)$ . Suppose that we want to constrain  $f(x)$  over interval  $[a, b]$ , where  $a < b$ , and the interval  $[a, b]$  is a subset of the original support of  $f(x)$ . Note that it is possible that  $a = -\infty$  or  $b = +\infty$ . Constraining  $f(x)$  in this manner will result in a new random variable,  $X|a \leq X \leq b$ . That is, given that  $X$  is contained in  $[a, b]$ ,  $X$  is the random variable. Random variables have a probability distribution. The question is what is the probability distribution of this new random variable and how can we generate from it.

This new random variable is governed by the conditional distribution of  $X$ , given that  $a \leq X \leq b$  and has the following form:

$$f(x|a \leq X \leq b) = f^*(x) = \begin{cases} \frac{g(x)}{F(b)-F(a)} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \quad (2.30)$$

where

$$g(x) = \begin{cases} f(x) & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \quad (2.31)$$

Note that  $g(x)$  is not a probability density. To convert it to a density, find its area and divide by its area. The area of  $g(x)$  is

$$\begin{aligned} \int_{-\infty}^{+\infty} g(x)dx &= \int_{-\infty}^a g(x)dx + \int_a^b g(x)dx + \int_b^{+\infty} g(x)dx \\ &= \int_{-\infty}^a 0 dx + \int_a^b f(x)dx + \int_b^{+\infty} 0 dx \\ &= \int_a^b f(x)dx = F(b) - F(a) \end{aligned}$$

Thus,  $f^*(x)$  is simply a “reweighting” of  $f(x)$ . The CDF of  $f^*(x)$  is

$$F^*(x) = \begin{cases} 0 & \text{if } x < a \\ \frac{F(x)-F(a)}{F(b)-F(a)} & a \leq x \leq b \\ 0 & \text{if } b < x \end{cases} \quad (2.32)$$

This leads to a straightforward algorithm for generating from  $f^*(x)$  as presented in Exhibit 2.13.

---

### Exhibit 2.13 Truncated Distribution Algorithm

---

- 1: Generate  $u \sim U(0, 1)$
  - 2:  $W = F(a) + (F(b) - F(a))u$
  - 3:  $X = F^{-1}(W)$
  - 4: RETURN  $X$
-

Lines 1 and 2 of the algorithm generate a random variable  $W$  that is uniformly distributed on  $(F(a), F(b))$ . Then, that value is used within the original distribution's inverse CDF function to generate  $X$ , given that  $a \leq X \leq b$ .

### ■ EXAMPLE 2.14 Truncated Distribution Example

Suppose  $X$  represents the distance between two cracks in highway. Suppose that  $X$  has an exponential distribution with a mean of 10 meters. Generate a distance restricted between 3 and 6 meters using the pseudorandom number 0.23.

The CDF of the exponential distribution with mean 10 is

$$F(x) = 1 - e^{-x/10}$$

Therefore,  $F(3) = 1 - \exp(-3/10) = 0.259$  and  $F(6) = 0.451$ . The exponential distribution has inverse CDF

$$F^{-1}(u) = \frac{-1}{\lambda} \ln(1-u)$$

First, we generate a random number uniformly distributed between  $F(3)$  and  $F(6)$  using  $u = 0.23$ :

$$W = 0.259 + (0.451 - 0.259) \times 0.23 = 0.3032$$

Therefore, to generate the distance, we have

$$X = -10 \times \ln(1 - 0.3032) = 3.612$$

Lastly, we discuss shifted distributions. Suppose  $X$  has a given distribution  $f(x)$ , then the distribution of  $X + \delta$  is termed the shifted distribution and is specified by  $g(x) = f(x - \delta)$ . It is easy to generate from a shifted distribution, simply generate  $X$  according to  $F(x)$  and then add  $\delta$ .

### ■ EXAMPLE 2.15 Shifted Distribution Example

Suppose  $X$  represents the time taken to set up a machine for production. From past time studies, we know that it cannot take less than 5.5 minutes to prepare for the setup and that the time after the 5.5 minutes is random with a Weibull distribution with shape parameter  $\alpha = 5$  and scale parameter  $\beta = 3$ . Using a pseudorandom number of  $u = 0.73$ , generate a value for the time to perform the setup.

The Weibull distribution has a closed-form CDF:

$$F(x) = 1 - e^{(-x/\beta)^\alpha} \quad (2.33)$$

Thus, the inverse CDF function is

$$F^{-1}(u) = \beta[-\ln(1-u)]^{1/\alpha} \quad (2.34)$$

Therefore, to generate the setup time, we have

$$5.5 + 5[-\ln(1 - 0.73)]^{1/3} = 5.47$$

## 2.6 SUMMARY

This chapter covered a number of important concepts used within simulation including

- generating random numbers
- testing random numbers
- generating random variates and processes.

These topics provide a solid foundation for modeling random components within your simulation models. Not only should you now understand how random numbers are generated but you should also know how to transform those numbers to allow the generation from a wide variety of probability distributions. With this understanding, you are now ready to develop useful simulation models involving random components. Chapter 3 will present how to add randomness to your spreadsheet modeling.

## EXERCISES

- 2.1 The sequence of random numbers generated from a given seed is called a random number (a)\_\_\_\_\_.
- 2.2 State three major methods of generating random variables from any distribution. (a)\_\_\_\_\_. (b)\_\_\_\_\_. (c)\_\_\_\_\_.
- 2.3 Consider the multiplicative congruential generator with  $a = 13, m = 64$ , and seeds  $X_0 = 1, 2, 3, 4$ .
  - (a) Does this generator achieve its maximum period for these parameters? Use Theorem 2.1 to justify your answer.
  - (b) Generate a period's worth of uniform random variables from each of the supplied seeds.
- 2.4 Consider the multiplicative congruential generator with  $a = 11, m = 64$ , and seeds  $X_0 = 1, 2, 3, 4$ .
  - (a) Does this generator achieve its maximum period for these parameters? Use Theorem 2.1 to justify your answer.
  - (b) Generate a period's worth of uniform random variables from each of the supplied seeds.
- 2.5 Analyze the following LCG:  $X_i = (11X_{i-1} + 5)(\text{mod } 16)$ ,  $X_0 = 1$  using Theorem 2.1.
  - (a) What is the maximum possible period length for this generator? Does this generator achieve the maximum possible period length? Justify your answer.

- (b) Generate two pseudorandom uniform numbers for this generator.
- 2.6 Analyze the following LCG generator:  $X_i = (13X_{i-1} + 13)(\text{mod } (16))$ ,  $X_0 = 37$  using Theorem 2.1.
- What is the maximum possible period length for this generator? Does this generator achieve the maximum possible period length? Justify your answer.
  - Generate two pseudorandom uniform numbers for this generator.
- 2.7 Analyze the following LCG generator:  $X_i = (4X_{i-1} + 3)(\text{mod } (16))$ ,  $X_0 = 11$  using Theorem 2.1.
- What is the maximum possible period length for this generator? Does this generator achieve the maximum possible period length? Justify your answer.
  - Generate two pseudorandom uniform numbers for this generator.
- 2.8 Analyze the following LCG generator:  $X_i = (8X_{i-1} + 1)(\text{mod } (10))$ ,  $X_0 = 3$  using Theorem 2.1.
- What is the maximum possible period length for this generator? Does this generator achieve the maximum possible period length? Justify your answer.
  - Generate two pseudorandom uniform numbers for this generator.
- 2.9 The following results are from a random sample of 100 uniform(0,1) numbers.

$n$	100
$\bar{x}$	0.4615
$s$	0.2915
Minimum	0.0102
First quartile	0.1841
median	0.4609
Third quartile	0.7039
Maximum	0.9687
$D^+$	0.090733
$D^-$	0.080733

- Form a 95% confidence interval for the mean. State any assumptions you need in order to make this confidence interval.
  - What would you conclude based on the K-S test results at the  $\alpha = 0.01$  level? Justify your answer using statistics.
- 2.10 Consider the following sequence of (0,1) random numbers:

0.943	0.398	0.372	0.943	0.204	0.794
0.498	0.528	0.272	0.899	0.294	0.156
0.102	0.057	0.409	0.398	0.400	0.997

Test if the sequence is distributed  $U(0, 1)$  using both a K-S test and a chi-squared test.

2.11 Consider the following set of pseudorandom numbers.

0.2379	0.7551	0.2989	0.247	0.3237
0.2972	0.8469	0.4566	0.6146	0.6723
0.9496	0.2268	0.8699	0.9084	0.5649
0.3045	0.6964	0.1709	0.3387	0.9804
0.1246	0.842	0.6557	0.9672	0.3356
0.3525	0.8075	0.9462	0.9583	0.3807
0.1489	0.5480	0.9537	0.9376	0.8364
0.5095	0.4047	0.9058	0.3795	0.6242
0.5195	0.6545	0.1117	0.3258	0.8589
0.6536	0.3427	0.6653	0.7864	0.5824

- (a) Test the hypothesis that these numbers are drawn from a  $U(0, 1)$  at a 95% confidence level using the chi-squared goodness-of-fit test using 10 intervals.
- (b) Test the hypothesis that these numbers are drawn from a  $U(0, 1)$  at a 95% confidence level using K-S test.
- (c) Test the hypothesis that these numbers are uniformly distributed within the unit square,  $\{(x, y) : x \in (0, 1), y \in (0, 1)\}$  using the 2D chi-Squared test at a 95% confidence level. Use four intervals for each of the dimensions.
- (d) Test the hypothesis that these numbers have a lag-1 correlation of zero. Make an autocorrelation plot of the numbers.
- 2.12 Consider the following discrete distribution of the random variable  $X$  whose PMF is  $p(x)$ .

$x$	0	1	2	3	4
$p(x)$	0.3	0.2	0.2	0.1	0.2

- (a) Determine the CDF  $F(x)$  for the random variable,  $X$ .
- (b) Create a graphical summary of the CDF. See Example 2.9.
- (c) Create a lookup table that can be used to determine a sample from the discrete distribution,  $p(x)$ . See Example 2.9.
- (d) Generate three values of  $X$  using the sequence of (0,1) random numbers in Exercise 2.10 (starting with the first row, reading across).
- 2.13 Consider the following uniformly distributed random numbers:

$U_1$	$U_2$	$U_3$	$U_4$	$U_5$	$U_6$	$U_7$	$U_8$
0.9396	0.1694	0.7487	0.3830	0.5137	0.0083	0.6028	0.8727

- (a) Generate an exponentially distributed random number with a mean of 10 using the first random number.
- (b) Generate a random variate from a (12, 22) discrete uniform distribution using the second random number.

2.14 Consider the following uniformly distributed random numbers:

$U_1$	$U_2$	$U_3$	$U_4$	$U_5$	$U_6$	$U_7$	$U_8$
0.9559	0.5814	0.6534	0.5548	0.5330	0.5219	0.2839	0.3734

- (a) Generate a uniformly distributed random number with a minimum of 12 and a maximum of 22 using  $U_8$ .
- (b) Generate one random variate from an Erlang( $r = 2, \beta = 3$ ) distribution using  $U_1$  and  $U_2$ .
- (c) The demand for magazines on a given day follows the following probability distribution

$x$	40	50	60	70	80
$P(X = x)$	0.44	0.22	0.16	0.12	0.06

Using the supplied random numbers for this problem starting at  $U_1$ , generate four random variates from this distribution.

- 2.15 Suppose that customers arrive at an ATM via a Poisson process with mean 7 per hour. Determine the arrival time of the first six customers using the data given in Exercise 2.10 (starting with the first row). Use the inverse transformation method.
- 2.16 The demand,  $D$ , for parts at a repair bench per day can be described by the following discrete PMF:

$D$	0	1	2
$p(D)$	0.3	0.2	0.5

Generate the demand for the first 4 days using the sequence of (0,1) random numbers in Exercise 2.10 (starting with the first row).

- 2.17 The service times for an automated storage and a retrieval system has a shifted exponential distribution. It is known that it takes a minimum of 15 seconds for any retrieval. The parameter of the exponential distribution is  $\lambda = 45$ . Using the sequence of (0,1) random numbers in Exercise 2.10 (starting with the first row), generate two service times for this situation.
- 2.18 The time to failure for a computer printer fan has a Weibull distribution with shape parameter  $\alpha = 2$  and scale parameter  $\beta = 3$ . Using the sequence of (0,1) random numbers in Exercise 2.10 (starting with the first row), generate two failure times for this situation.
- 2.19 The time to failure for a computer printer fan has a Weibull distribution with shape parameter  $\alpha = 2$  and scale parameter  $\beta = 3$ . Testing has indicated that the distribution is limited to the range from 1.5 to 4.5. Using the sequence of (0,1) random numbers in Exercise 2.10 (starting with the first row), generate two failure times for this truncated distribution.
- 2.20 The interest rate for a capital project is unknown. An accountant has estimated that the minimum interest rate will be between 2% and 5% within the next year. The

accountant believes that any interest rate in this range is equally likely. You are tasked with generating interest rates for a cash flow analysis of the project. Using the sequence of (0,1) random numbers in Exercise 2.10 (starting with the first row), generate two independent interest rate values for this situation.

- 2.21 Customers arrive at a service location according to a Poisson distribution with mean 10 per hour. The installation has two servers. Experience shows that 60% of the arriving customers prefer the first server. By using the first row of (0,1) random numbers given in Exercise 2.10, determine the arrival times of the first three customers at each server.
- 2.22 Consider the triangular distribution

$$F(x) = \begin{cases} 0 & x < a \\ \frac{(x-a)^2}{(b-a)(c-a)} & a \leq x \leq c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)} & c < x \leq b \\ 1 & b < x \end{cases}$$

- (a) Derive an inverse transform algorithm for this distribution.
- (b) Using the first row of random numbers from Exercise 2.10, generate five random numbers from the triangular distribution with  $a = 2$ ,  $c = 5$ , and  $b = 10$ .
- 2.23 Consider the following PDF:

$$f(x) = \begin{cases} \frac{3x^2}{2} & -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

- (a) Derive an inverse transform algorithm for this distribution.
- (b) Using the first row of random numbers from Exercise 2.10, generate two random numbers using your algorithm.
- 2.24 Consider the following PDF:

$$f(x) = \begin{cases} 0.5x - 1 & 2 \leq x \leq 4 \\ 0 & \text{otherwise} \end{cases}$$

- (a) Derive an inverse transform algorithm for this distribution.
- (b) Using the first row of random numbers from Exercise 2.10, generate two random numbers using your algorithm.
- 2.25 Consider the following PDF:

$$f(x) = \begin{cases} \frac{2x}{25} & 0 \leq x \leq 5 \\ 0 & \text{otherwise} \end{cases}$$

- (a) Derive an inverse transform algorithm for this distribution.  
 (b) Using the first row of random numbers from Exercise 2.10, generate two random numbers using your algorithm.

2.26 Consider the following PDF:

$$f(x) = \begin{cases} \frac{2}{x^3} & x > 1 \\ 0 & x \leq 1 \end{cases}$$

- (a) Derive an inverse transform algorithm for this distribution.  
 (b) Using the first row of random numbers from Exercise 2.10, generate two random numbers using your algorithm.

2.27 The times to failure for an automated production process have been found to be randomly distributed according to a Rayleigh distribution:

$$f(x) = \begin{cases} 2\beta^{-2}xe^{-(x/\beta)^2} & x > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.35)$$

- (a) Derive an inverse transform algorithm for generating random variables from this distribution.  
 (b) Using the first row of random numbers from Exercise 2.10, generate five random numbers from your algorithm with  $\beta = 2$ .

2.28 Using the first two rows of random numbers from Exercise 2.10, generate five random numbers from the negative binomial distribution with parameters ( $r = 4, p = 0.4$ ) using:  
 (a) the convolution method  
 (b) the number of Bernoulli trials to get four successes.

2.29 Suppose that the processing time for a job consists of two distributions. There is a 30% chance that the processing time is lognormally distributed with a mean of 20 minutes and a standard deviation of 2 minutes, and a 70% chance that the time is uniformly distributed between 10 and 20 minutes. Using the first row of random numbers from Exercise 2.10, generate two job processing times. Hint:  $X \sim \ln(\mu, \sigma^2)$  if and only if  $\ln(X) \sim N(\mu, \sigma^2)$ . Also, note that

$$\mathbb{E}[X] = e^{\mu + \sigma^2/2}$$

$$\text{Var}[X] = e^{2\mu + \sigma^2}(e^{\sigma^2} - 1)$$

2.30 Suppose that the service time for a patient consists of two distributions. There is a 25% chance that the service time is uniformly distributed with minimum of 20 minutes and a maximum of 25 minutes, and a 75% chance that the time is distributed according to a Weibull distribution with shape of 2 and a scale of 4.5. Using the first row of random numbers from Exercise 2.10, generate the service time for two patients.

- 2.31 If  $Z \sim N(0, 1)$ , and  $Y = \sum_{i=1}^k Z_i^2$  then  $Y \sim \chi_k^2$ , where  $\chi_k^2$  is a chi-squared random variable with  $k$  degrees of freedom. Using the first two rows of random numbers from Exercise 2.10, generate two  $\chi_5^2$  random variates.
- 2.32 In the (a)\_\_\_\_\_ technique for generating random variates, you want the (b)\_\_\_\_\_ function to be as close as possible to the distribution function that you want to generate from in order to ensure that the (c)\_\_\_\_\_ is as high as possible, thereby improving the efficiency of the algorithm.
- 2.33 Prove that the acceptance–rejection method for continuous random variables is valid by showing that for any  $x$ ,

$$P\{X \leq x\} = \int_{-\infty}^x f(y)dy$$

Hint: Let  $E$  be the event that the acceptance occurs and use conditional probability.

- 2.34 Consider the following PDF:

$$f(x) = \begin{cases} \frac{3x^2}{2} & -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

- (a) Derive an acceptance–rejection algorithm for this distribution.  
 (b) Using the first row of random numbers from Exercise 2.10, generate two random numbers using your algorithm.
- 2.35 This problem is based on Cheng [1977], see also Ahrens and Dieter [1972]. Consider the gamma distribution

$$f(x) = \beta^{-\alpha} x^{\alpha-1} \frac{e^{-x/\beta}}{\Gamma(\alpha)}$$

where  $x > 0$  and  $\alpha > 0$  is the shape parameter and  $\beta > 0$  is the scale parameter. In the case where  $\alpha$  is a positive integer, the distribution reduces to the Erlang distribution and  $\alpha = 1$  produces the negative exponential distribution. Acceptance–rejection techniques can be applied to the cases of  $0 < \alpha < 1$  and  $\alpha > 1$ . For the case of  $0 < \alpha < 1$ , see Ahrens and Dieter [1972]. For the case of  $\alpha > 1$ , Cheng [1977] proposed the following majorizing function:

$$g(x) = \left[ \frac{4\alpha^\alpha e^{-\alpha}}{a\Gamma(\alpha)} \right] h(x)$$

where  $a = \sqrt{(2\alpha - 1)}$ ,  $b = \alpha^a$ , and  $h(x)$  is the resulting probability distribution function when converting  $g(x)$  to a density function:

$$h(x) = ab \frac{x^{\alpha-1}}{(b + x^\alpha)^2} \quad \text{for } x > 0$$

- (a) Develop an inverse transform algorithm for generating from  $h(x)$ .  
 (b) Using the first two rows of random numbers from Exercise 2.10, generate two random variates from a gamma distribution with parameters  $\alpha = 2$  and  $\beta = 10$  via the acceptance–rejection method.

- 2.36 Parts arrive to a machine center with three drill presses according to a Poisson distribution with mean  $\lambda$ . The arriving customers are assigned to one of the three drill presses randomly according to the respective probabilities  $p_1, p_2$ , and  $p_3$  where  $p_1 + p_2 + p_3 = 1$  and  $p_i > 0$  for  $i = 1, 2, 3$ . What is the distribution of the interarrival times to each drill press? Specify the parameters of the distribution.

Suppose that  $p_1, p_2$ , and  $p_3$  are equal to 0.25, 0.45, and 0.3, respectively, and that  $\lambda$  is equal to 12 per minute. Generate the first three arrival times using numbers from the first row of random numbers from Exercise 2.10.



---

# 3

---

## SPREADSHEET SIMULATION

### LEARNING OBJECTIVES

- To be able to generate random numbers within a spreadsheet environment.
- To be able to generate random variables with various distributional models within a spreadsheet.
- To be able to perform simple Monte-Carlo simulation in a spreadsheet environment.
- To be able to compute statistical quantities for a Monte-Carlo simulation.
- To be able to determine the sample size for a Monte-Carlo simulation.

### 3.1 SIMULATION IN A SPREADSHEET ENVIRONMENT

In this chapter, you will learn how to perform basic simulation methods within a spreadsheet environment. The immediacy of a spreadsheet environment will facilitate the understanding of many of the methods discussed in Chapter 2. In addition, you will learn about some of the functions that are available within spreadsheets that can make simulation a useful modeling tool for simple to moderately complex spreadsheet simulations. This introduction gives you enough information to begin to appreciate the more advanced spreadsheet add-in tools for performing simulations within a spreadsheet environment.

In the next section, we put the theory of Chapter 2 into practice using spreadsheets.

## 3.2 USEFUL SPREADSHEET FUNCTIONS AND METHODS

This section discusses how to implement a few of the random number generation techniques mentioned in Chapter 2 within a spreadsheet. The purpose is to give you look at how these concepts can be put into practice.

### 3.2.1 Using RAND() and RANDBETWEEN()

The spreadsheet *SimulatingRandomnessInExcel.xlsxm* has the examples that are discussed in this section. The RAND() function in Excel™ will generate a random number uniformly distributed on the interval from (0,1). From this random number, many other random numbers can be generated using the methods discussed in Chapter 2.

The RAND() function is an “active” function. That is, each time the worksheet is calculated, a new value from RAND() will be returned. For example, a worksheet will be recalculated if a cell is evaluated. As you will see, this can be inconvenient when trying to work with large spreadsheets that simulate many random variables.

Excel™ also has the worksheet function RANDBETWEEN( $a,b$ ) which will generate a random integer between  $[a,b]$ . This function is also an active function. To turn off automatic calculation, you can set the calculation option for the spreadsheet to manual as illustrated in Figure 3.1. If you do turn off automatic calculation, you can use F9 to cause the spreadsheet to recalculate.

If you are following along to reproduce the spreadsheets shown in this section, you should keep in mind that your random numbers may be different from that shown in the screenshots. This is because there is no way to set the seed for the RAND() function (Figure 3.2). By using the RAND() and RANDBETWEEN( $a,b$ ) functions, a great deal of useful work can be performed (Figure 3.3).

To get a Uniform( $a,b$ ) number, use the cell formula

$$= a + (b - a) * \text{RAND()}$$

$(b - a) * \text{RAND}()$  generates a number between 0 and  $(b - a)$ . Then, by adding  $a$ , the number is translated to the appropriate range on the  $x$ -axis. This is the inverse transform method.

The RANDBETWEEN() function generates a discrete uniform over the range from  $a$  to  $b$ . If the RANDBETWEEN() function was not available, you can use

$$= a + \text{INT}((b - a + 1) * \text{RAND}())$$

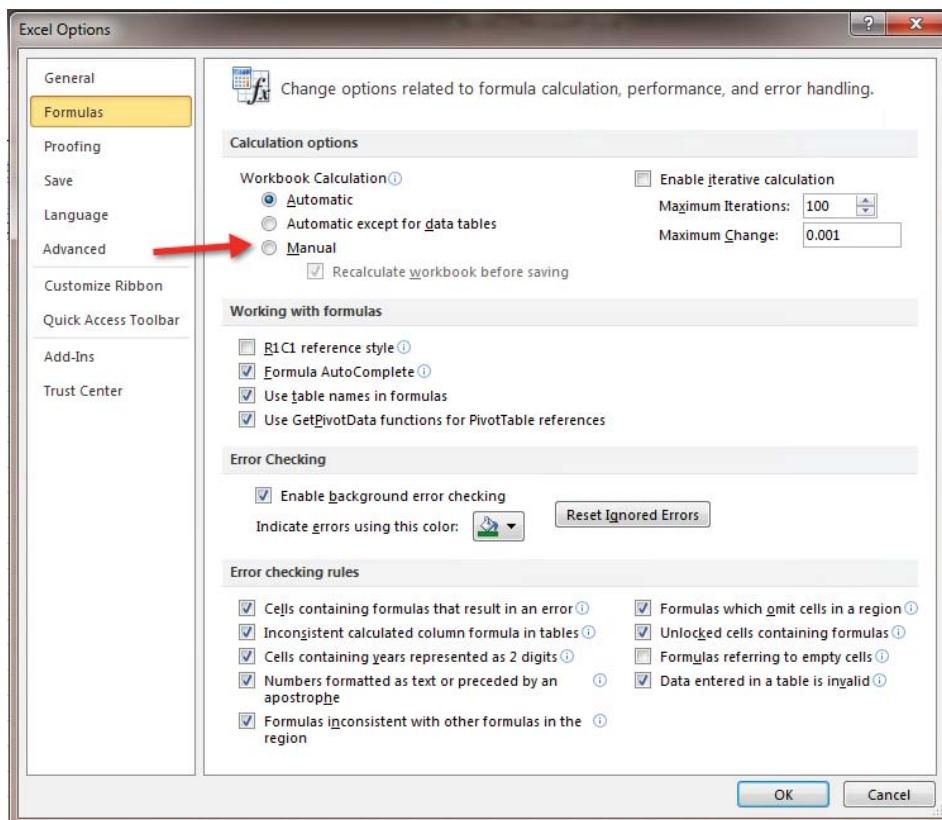
A Bernoulli( $p$ ) random variable, where  $p$  is the probability of success, can be implemented by using the spreadsheet’s IF() function in combination with the RAND() function.

$$= \text{IF}(\text{RAND}() \leq p, 1, 0)$$

To implement the generation of binomial random variables, you can easily sum the cells containing the Bernoulli trials, which implements the convolution method. The shifted geometric distribution can be implemented with the cell formula

$$= 1 + \text{INT}(\ln(1 - \text{RAND}) / \ln(1 - p))$$

This implements the inverse transform algorithm for the geometric distribution as presented in Exhibit 2.7.



**Figure 3.1** Setting the worksheet to manual calculation.

	B5	f	=RAND()	
	A	B	C	D
1		min=	2	
2		max=	10	
3				
4		U(0,1)	DU(min,max)	U(min,max)
5		1 0.478231133	5 9.229413517	
6		2 0.786127707	3 9.153368806	
7		3 0.158092971	2 8.131687769	
8		4 0.605374938	8 4.690445215	
9		5 0.138187052	10 8.877661769	
10		6 0.909570015	5 3.748755936	
11		7 0.700547878	4 7.361288708	
12		8 0.539418158	2 2.451534615	
13		9 0.543234406	4 5.705660412	
14		10 0.508808182	5 5.739570912	

**Figure 3.2** Use of RAND() function.

	A	B	C
1		min=	2
2		max=	10
3			
4	U(0,1)	DU(min,max)	U(min,max)
5	=RAND()	=RANDBETWEEN(\$C\$1,\$C\$2)	=\$C\$1+
6	=RAND()	=RANDBETWEEN(\$C\$1,\$C\$2)	=\$C\$1+
7	=RAND()	=RANDBETWEEN(\$C\$1,\$C\$2)	=\$C\$1+
8	=RAND()	=RANDBETWEEN(\$C\$1,\$C\$2)	=\$C\$1+
9	=RAND()	=RANDBETWEEN(\$C\$1,\$C\$2)	=\$C\$1+
10	=RAND()	=RANDBETWEEN(\$C\$1,\$C\$2)	=\$C\$1+
11	=RAND()	=RANDBETWEEN(\$C\$1,\$C\$2)	=\$C\$1+
12	=RAND()	=RANDBETWEEN(\$C\$1,\$C\$2)	=\$C\$1+
13	=RAND()	=RANDBETWEEN(\$C\$1,\$C\$2)	=\$C\$1+
14	=RAND()	=RANDBETWEEN(\$C\$1,\$C\$2)	=\$C\$1+

Figure 3.3 Use of RAND() and RANDBETWEEN() functions.

A number of continuous distributions are easy to implement within a spreadsheet due to the fact that the inverse cumulative distribution function (CDF) function for the distribution is available. In these formulas, you replace the argument requesting a probability with the RAND() function. For example, to simulate a normal random variable, you use the cell formula:

$$= \text{NORM.INV}(\text{RAND}(), \mu, \sigma)$$

where  $\mu$  and  $\sigma$  are the mean and the standard deviation for the desired normal variates, respectively. Figure 3.4 illustrates the various random variables that are easy to implement in a spreadsheet. This comes from the sheet labeled Distributions in this chapter's example spreadsheet.

### 3.2.2 Using VLOOKUP()

The VLOOKUP cell function provides an easy way to implement the inverse transform method for discrete distributions. Let us take a look at an example and implement it with the VLOOKUP function. Let  $X$  be a discrete random variable, with  $X \in \{1, 2, 3, 4\}$  and the probability mass function (PMF) and CDF as given in Table 3.1.

This is the same distribution as that used in Example 2.9. The key to use the VLOOKUP function is to make a look-up table that implements Equation (2.24):

$$F^{-1}(u) = \begin{cases} 1 & \text{if } 0.0 \leq u \leq 0.4 \\ 2 & \text{if } 0.4 < u \leq 0.7 \\ 3 & \text{if } 0.7 < u \leq 0.9 \\ 4 & \text{if } 0.9 < u \leq 1.0 \end{cases}$$

To create the table for VLOOKUP, you should set up your spreadsheet as indicated in Figure 3.5.

- The column PMF is the PMF values.

	A	B	C	D	E
1					Formula
2	Uniform(0,1)			0.845009617	=RAND()
3		min	max		
4	Discrete Uniform(min,max)	3	10	8	=RANDBETWEEN(B5,C5)
5		min	max		
6	Uniform(min,max)	4	20	18.66724518	=B7+(C7-B7)*RAND()
7			p		
8	Bernoulli(p)		0.7	1	=IF(RAND()<=C9,1,0)
9			p		
10	Geometric(p)		0.3	2	=1+INT(LN(1-RAND())/LN(1-C11))
11			mean		
12	Exponential(mean)		5	4.359013352	=-C13*LN(1-RAND())
13		scale	shape		
14	Weibull(scale, shape)	5	3	3.975532185	=B15*POWER(-LN(1-RAND()),1/C15)
15		alpha1	alpha2		
16	Beta(alpha1, alpha2)	5	1.5	0.807041407	=BETAINV(RAND(),B17,C17)
17			DF		
18	ChiSquare		5	14.12091438	=CHIINV(RAND(),C19)
19		scale	shape		
20	Gamma(scale,shape)	2	10	19.06188666	=GAMMAINV(RAND(),C21,B21)
21		mean	std dev		
22		5	3		
23	Lognormal(mean, std dev)	1.455695563	0.554513	5.012218975	=LOGINV(RAND(),B24,C24)
24		mean	std dev		
25	Normal(mean, std dev)	10	2	10.50135199	=NORMINV(RAND(),B25,C25)

**Figure 3.4** Generating from distributions within a spreadsheet.**TABLE 3.1 PMF and CDF for VLOOKUP Example**

$x_i$	1	2	3	4
$f(x_i)$	0.4	0.3	0.2	0.1
$F(x_i)$	0.4	0.7	0.9	1.0

A8 ▾ fx =VLOOKUP(RAND(),B3:D6,3)				
A	B	C	D	E
1				
2	PMF	LR	CDF	X
3	0.4	0	0.4	1
4	0.3	0.4	0.7	2
5	0.2	0.7	0.9	3
6	0.1	0.9	1	4
7				
8	2	=VLOOKUP(RAND(),B3:D6,3)		

**Figure 3.5** Setting up the LOOKUP table.

- The column LR is the lower limit on the range for the value. For example, if the random number  $U$  falls between  $(0, 0.4)$ , then the X is set to 1 where the CDF column specifies the upper range.
- The column X should contain the possible values for the random variable.

Now you can use the following:

$$= VLOOKUP(RAND(), \text{CellRange}, 3)$$

where Cell Range is the range for the LR-X columns. In the example, Cell Range is B3:D6. RAND() generates a number between 0 and 1. This number is looked up in the table's first column (of Cell Range) and the corresponding value from the third column (the 3 in the formula) is returned. This works because of the way in which VLOOKUP does its search. This general scheme will work for any discrete random variable. In fact, the X column does not have to have numbers. It could contain text and the VLOOKUP function will randomly select the value with the appropriate probabilities.

The following section illustrates how to generate a random sample using a data table.

### 3.2.3 Using Data Tables to Repeatedly Sample

Figure 3.5 illustrates how to generate one random number from the provided PMF using the VLOOKUP() function. In order to generate additional observations from the distribution, we can do two things: (i) just drag the formula into other cells or (ii) use the data table functionality of the spreadsheet environment. The previous examples used the first method and dragged the formula (i.e., copied the formula). This can become tedious. The second method of using data tables is more general and facilitates varying parameters in the simulation. This section illustrates the basic methodology. Section 3.3 illustrates the use of data tables in a more complete manner.

Suppose that we want to repeat the generation from the PMF 20 times. This can be done using data tables but requires a little setup of the spreadsheet. A data table represents a range of cells that can allow the varying of one or two variables within a cell formula. This allows for multiple results to be easily computed in one operation. Data tables allow for easy tabulation of the results on one sheet. In the case of a column-oriented data table, the values in the left column of the data table are substituted into the cell designated as the column input cell and the output cells (top row of the data table) are recomputed. Then, the spreadsheet is recalculated and the values in the top row of the data table are returned for each of the input values.

The following process describes how to set up a data table to replicate a simulation model. Figure 3.6 illustrates the process in a spreadsheet.

1. Make a column to count the replications. Column F is used in this example.
2. Make a cell that you want to be repeated. Cell A8 is used in this example.
3. Tie the data table to the cell that needs repeating by setting the cell above and to the right of the first replication equal to the cell to be repeated. In this example, Cell G4 is set equal to cell A8.
4. Select the cells that are to form the data table. In this example, the cells in the range F4:G24 are the required cells.
5. From the spreadsheet menu invoke the data table functionality. This is Data  $\Rightarrow$  What-IF  $\Rightarrow$  Data Table. A confusing dialog box will appear asking for a row input cell and a column input cell. Since the table is column oriented, we only need the column input cell.
6. Enter any blank cell in the column input cell. Cell F4 was chosen as the input cell in this example. Leave the row input cell blank. Select the OK button.

	A	B	C	D	E	F	G	
1								
2	PMF	LR	CDF	X				
3	0.4	0	0.4	1				Data Table for Replications
4	0.3	0.4	0.7	2				Replication Observation
5	0.2	0.7	0.9	3				2
6	0.1	0.9	1	4				1 1
7								2 2
8								3 1
9								4 1
10								5 3
11								6 1
12								7 3
13								8 3
14								9 3
15								10 2
16								11 1
17								12 2
18								13 2
19								14 1
20								15 3
21								16 1
22								17 1
23								18 1
24								19 2
25								20 1
26								
27								

Data Table Example

- 1) Make a column to count the replications
  - a) Column F here
- 2) Make a cell that you want repeated
  - a) Cell A8 will be repeated here
- 3) Tie the data table to the cell by setting the cell above and to the right of the 1st replication equal to the cell to be repeated
  - a) Cell G4 here is set to cell A8
- 4) Select the cells to form the data table
  - a) Select cells F4:G24
- 5) Select Data, What-If, Data Table
- 6) Enter any blank cell in for the Column input cell
  - a) Here cell F4 is used
  - b) Leave the Row input cell blank

Figure 3.6 Spreadsheet results from data table.

In the case of simulation, data tables can be tricked into repeating a formula many times by selecting a blank input cell. Since the input cell is blank, nothing is substituted; however, the spreadsheet is recalculated. This recalculation results in a new random number being generated and then returned in the data rows. This results in multiple samples being generated, one for each row of the data table.

Very large data tables can require a lot of computing time. By turning off the automatic calculation, as per Figure 3.1, working with the spreadsheet becomes easier. The option “Automatic Except Tables” will cause the spreadsheet to operate as normal, except that the data tables will not automatically recalculate. You will need to manually recalculate the spreadsheet to get the data table results.

### 3.2.4 Using VBA

If spreadsheet cell formulas are inadequate for your work, you can always implement random variate generation algorithms in a VBA function. Listings 3.1 and 3.2 illustrate how the functions use the VBA function Rnd() to generate a  $U(0,1)$  random number and then implement the algorithms discussed in the text. These functions can be found in the VBA module named Random within the example spreadsheet.

The VBA function bernoulliRV( $p$  As Double) takes in the success probability as a parameter. The function Rnd() is used to generate a  $U(0,1)$  value, which is compared to  $1 - p$ . If the comparison is true, the result of the function is 0; otherwise, it is 1. This is the inverse transform algorithm for Bernoulli random variables.

The VBA function binomialRV( $p$  As Double,  $n$  as Integer) takes in the success probability and the number of trials as inputs and returns a binomial random variable using the convolution algorithm. The functions expo() and weibulRV() use the inverse transform technique for the exponential distribution and the Weibull distribution, respectively. Because

these functions are in a VBA module for the spreadsheet, they can be used within any cell in the spreadsheet, just like any of VBA's built in functions.

### **Listing 3.1 VBA Functions for Bernoulli and Binomial Random Variables**

```
Public Function bernoulliRV(p As Double) As Integer
    If (Rnd() < (1 - p)) Then
        bernoulliRV = 0
    Else
        bernoulliRV = 1
    End If
End Function

Public Function binomialRV(p As Double, n As Integer) As Integer
    Dim X As Integer
    Dim i As Integer
    X = 0
    For i = 1 To n
        X = X + bernoulliRV(p)
    Next i
    binomialRV = X
End Function
```

### **Listing 3.2 VBA Functions for Exponential and Weibull Random Variables**

```
Public Function expo(mean As Double) As Double
    expo = -mean * Log(1 - Rnd())
End Function

Public Function weibullRV(scaleB As Double, shapeA As Double) As Double
    weibullRV = scaleB * (-Log(1 - Rnd())) ^ (1 / shapeA)
End Function
```

The following section illustrates how to perform simple spreadsheet simulations using the concepts discussed in this section.

### **3.3 EXAMPLE SPREADSHEET SIMULATIONS**

A spreadsheet can be readily used to implement a simulation analysis. This section illustrates how to simulate three classic problems using a spreadsheet. An important aspect of any simulation is that it is well organized and documented. This is especially important in a spreadsheet environment because spreadsheets are notorious about having embedded errors that are hard to find. Powell et al. [2008] and Rajalingham et al. [2000] discussed some of the quality control and error issues related to spreadsheet modeling. Before committing to use spreadsheets, it is important to understand their limitations as well as best practices for documenting them (see LeBlanc and Galbreth [2009]). In addition, you should be aware of some of the statistical problems/errors that are embedded with spreadsheet tools. Please see McCullough and Heiser [2008] and the references therein for more details on this serious

topic. With that said, we will use spreadsheets and the examples with an attempt to illustrate some useful practices for how to organize your spreadsheet simulations.

### 3.3.1 Simple Monte-Carlo Integration

The term Monte-Carlo generally refers to the set of methods and techniques predicated on estimating quantities by repeatedly sampling from models/equations represented in a computer. As such, this terminology is somewhat synonymous with computer simulation itself. The term Monte-Carlo gets its origin from the Monte-Carlo casino in the Principality of Monaco, where gambling and games of chance are well known. There is no one Monte-Carlo method. Rather there is a collection of algorithms and techniques. In fact, the ideas of random number generation and random variate generation previously discussed form the foundation of Monte-Carlo methods.

For the purposes of this chapter, we limit the term Monte-Carlo methods to those techniques for generating and estimating the expected values of random variables, especially in regards to static simulation. In static simulation, the notion of time is relatively straightforward with respect to system dynamics. For a static simulation, time “ticks” in a regular pattern and at each “tick,” the state of the system changes (new observations are produced). This should be contrasted with dynamic simulation, where the state of the system evolves over time and the state changes at irregularly (randomly occurring) points in time. Dynamic simulation (specifically discrete-event simulation) will be the subject of subsequent chapters.

Here we will illustrate one of the fundamental uses of Monte-Carlo methods: estimating the area of a function. Suppose we have some function,  $g(x)$ , defined over the range  $a \leq x \leq b$  and we want to evaluate the integral:

$$\theta = \int_a^b g(x) dx \quad (3.1)$$

Monte-Carlo methods allow us to evaluate this integral by couching the problem as an estimation problem. It turns out that the problem can be translated into estimating the expected value of a well-chosen random variable. While a number of different choices for the random variable exist, we will pick one of the simplest for illustrative purposes. Define  $Y$  as follows with  $X \sim U(a, b)$ :

$$Y = (b - a)g(X) \quad (3.2)$$

Notice that  $Y$  is defined in terms of  $g(X)$ , which is also a random variable. Because a function of a random variable is also a random variable, this makes  $Y$  a random variable . Thus, the expectation of  $Y$  can be computed as follows:

$$E[Y] = E[(b - a)g(X)] = (b - a) E[g(X)] \quad (3.3)$$

Now, let us derive  $E[g(X)]$ . By definition,

$$E_X[g(X)] = \int_a^b g(x) f_X(x) dx$$

And, the probability density function for a  $X \sim U(a, b)$  random variable is

$$f_X(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

Therefore,

$$\mathbb{E}_X[g(X)] = \int_a^b g(x) f_X(x) dx = \int_a^b g(x) \frac{1}{b-a} dx$$

Substituting into Equation (3.3) yields

$$\begin{aligned} \mathbb{E}[Y] &= \mathbb{E}[(b-a)g(X)] = (b-a)\mathbb{E}[g(X)] \\ &= (b-a) \int_a^b g(x) \frac{1}{b-a} dx \\ &= \int_a^b g(x) dx = \theta \end{aligned}$$

Therefore, by estimating the expected value of  $Y$ , we can estimate the desired integral. From basic statistics, we know that a good estimator for  $\mathbb{E}[Y]$  is the sample average of observations of  $Y$ . Let  $Y_1, Y_2, \dots, Y_n$  be a random sample of observations of  $Y$ . Let  $X_i$  be the  $i$ th observation of  $X$ . Substituting each  $X_i$  into Equation (3.2) yields the  $i$ th observation of  $Y$ ,

$$Y_i = (b-a)g(X_i) \quad (3.4)$$

Then, the sample average of  $Y_1, Y_2, \dots, Y_n$  is

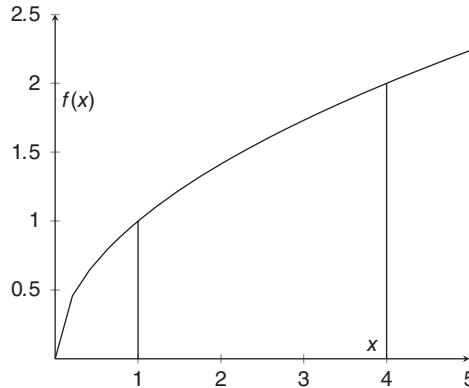
$$\begin{aligned} \bar{Y}(n) &= \frac{1}{n} \sum_{i=1}^n Y_i = \frac{1}{n} \sum_{i=1}^n (b-a)g(X_i) \\ &= (b-a) \frac{1}{n} \sum_{i=1}^n g(X_i) \end{aligned}$$

where  $X_i \sim U(a, b)$ . Thus, by simply generating  $X_i \sim U(a, b)$ , plugging the  $X_i$  into the function of interest,  $g(x)$ , taking the average over the values, and multiplying by  $(b-a)$ , we can estimate the integral. This works for any integral and it works for multidimensional integrals. While this discussion is based on a single-valued function, the theory scales to multidimensional integration through the use of multivariate distributions. Now, let us illustrate the estimation of the area under a curve using a spreadsheet.

### EXAMPLE 3.1 Estimating the Area Under a Curve

Suppose that we want to estimate the area under  $f(x) = x^{\frac{1}{2}}$  over the range from 1 to 4 as illustrated in Figure 3.1. That is, we want to evaluate the integral

$$\theta = \int_1^4 x^{\frac{1}{2}} dx = \frac{14}{3} = 4.6\bar{6} \quad (3.5)$$



According to the previously presented theory, we need to generate  $X_i \sim U(1, 4)$  and then compute  $\bar{Y}$ , where  $Y_i = (4 - 1)\sqrt{X_i} = 3\sqrt{X_i}$ . This can be readily accomplished using a spreadsheet. In addition, for this simple example, we can easily check if our Monte-Carlo approach is working because we know the true area.

The spreadsheet solution to this example can be found on the *Simulation MC* tab of the spreadsheet accompanying this chapter. Recall that the process is to generate  $X_i \sim U(a, b)$ ; plug the  $X_i$  into the function of interest,  $f(x)$ ; and take the average over the values and multiply by  $(b - a)$ . The process is very straightforward.

1. Use a column to hold the  $U(0, 1)$  random numbers (e.g., column B).
2. Use a column to hold the sample  $X_i \sim U(a, b)$  (e.g., column C).
3. Use a column to hold the sample  $f(X_i)$  (e.g., column D).
4. Use a column to hold the sample  $Y_i = (b - a)f(X_i)$  (e.g., column E).
5. Compute the average of the sample of  $Y_i$  via the `AVERAGE()` function (e.g., Cell E4).

Figure 3.7 illustrates how to set up a spreadsheet for this problem. Notice how the spreadsheet is arranged: inputs at the top, simulation outputs toward the bottom, and a text box that describes the spreadsheet implementation.

Example 3.1 estimates the area under a very simple function. In fact, we know from basic calculus that the area is  $\theta = 14/3 = 4.67$ . As can be seen in the solution, the result of the simulation (cell F4) is very close to the true value. The difference between the estimated result and the true value is due to *sampling error*. We will discuss this important topic in Section 3.4.

Even though this example is rather simple, the basic idea can be used for any complicated function and can be readily adapted for multidimensional integrals.

### 3.3.2 The Classic News Vendor Inventory Problem

The news vendor model is a classic inventory model that allows for the modeling of how much to order for a decision maker facing uncertain demand for an upcoming period of time. It takes on the news vendor moniker because of the context of selling newspapers at a newsstand. The vendor must anticipate how many papers to have on hand at the beginning

A	B	C	D	E	F	G	H	I	J
1			True Area =	4.6666667					
2	Lower Limit a =		1						
3	Upper Limit b =		4						
4			Average =	4.7421464					
5									
6	n	U(0,1)	X ~ U(a,b)	f(X)	Y=(b-a)f(X)				
7	1	0.997191473	3.9915744	1.9978925	5.9936775				
8	2	0.989081344	3.967244	1.9917942	5.9753825				
9	3	0.21772085	1.6531625	1.2857537	3.8572611				
10	4	0.661895436	2.9856863	1.7279139	5.1837416				
11	5	0.404142642	2.2124279	1.4874233	4.4622698				
12	6	0.296216657	1.88865	1.3742816	4.1228449				
13	7	0.613861009	2.841583	1.6856996	5.0570987				
14	8	0.37081048	2.1124314	1.4534206	4.3602618				
15	9	0.904623825	3.7138715	1.9271408	5.7814223				
16	10	0.99243608	3.9773082	1.994319	5.982957				
17	11	0.452834101	2.3585023	1.5357416	4.6072248				
18	12	0.174478216	1.5234346	1.2342749	3.7028248				
19	13	0.228178088	1.6845343	1.2978961	3.8936883				
20	14	0.035379788	1.1061394	1.0517316	3.1551948				
21	15	0.507718358	2.5231551	1.5884442	4.7653327				
22	16	0.983199878	3.9495996	1.98736	5.9620799				
23	17	0.689793097	3.0693793	1.7519644	5.2558932				
24	18	0.272101333	1.816304	1.3477032	4.0431097				
25	19	0.799774928	3.3993248	1.8437258	5.5311774				
26	20	0.034046554	1.1021397	1.0498284	3.1494852				
27									

**Simulating the Area**  
 Want to estimate the area under the curve:  
 $f(x) = \sqrt{x}$   
 over the range [1, 4]  
 The true area is  $1/3 = 4.6666$  (see cell E1).  
 1) column B holds values from rand() to generate U(0,1)  
 2) column C generates X ~ U(a, b) using inverse transform  
 $X = a + (b-a)*U$   
 e.g. row 7:  $=\$C\$2 + (\$C\$3-\$C\$2)*B7$   
 3) column D evaluates f(x) at the generated X  
 e.g. row 7:  $=SQRT(C7)$   
 4) column E evaluates Y = (b-a)\*f(X)  
 e.g. row 7:  $=(\$C\$3-\$C\$2)*D7$   
 5) cell E4 computes the average of the Y's  
 $=AVERAGE(E7:E26)$   
 Notice how close cell E4 is to cell E1

Figure 3.7 Spreadsheet Area Estimation Example

of a day so as to not run short or have papers left over. This idea can be generalized to other more interesting situations (e.g., air plane seats). Discussion of the news vendor model is often presented in elementary textbooks on inventory theory. The reader is referred to Muckstadt and Sapras [2010] for a discussion of this model.

The basic model is considered a single-period model; however, it can be extended to consider an analysis cover multiple (or infinite) future periods of demand. The representation of the costs within the modeling offers a number of variations.

Let  $D$  be a random variable representing the demand for the period. Let  $F(d) = P\{D \leq d\}$  be the CDF for the demand. Define  $G(q, D)$  as the profit at the end of the period when  $q$  units are ordered at the start of the period with  $D$  units of demand. The parameter  $q$  is the decision variable associated with this model. Depending on the value of  $q$  chosen by the news vendor, a profit or loss will occur.

There are two cases to consider  $D \geq q$  or  $D < q$ , that is, when demand is greater than or equal to the amount ordered at the start of the period and when demand is less than the amount ordered at the start of the period. If  $D \geq q$ , the news vendor runs out of items, and if  $D < q$ , the news vendor will have items left over.

In order to develop a profit model for this situation, we need to define some model parameters. In addition, we need to determine the amount that we might be short and the amount that we might have left over in order to determine the revenue or loss associated with a choice of  $q$ . Let  $c$  be the purchase cost. This is the cost that the news vendor pays the supplier for the item. Let  $s$  be the selling price. This is the amount that the news vendor charges customers. Let  $u$  be the salvage value (i.e., the amount per unit that can be received for the item after the planning period). For example, the salvage value can be the amount that the

news vendor gets when selling a leftover paper to a recycling center. We assume that selling price > purchase cost > salvage value.

Consider the context of a news vendor planning for the day. What is the possible amount sold? If  $D$  is the demand for the day and we start with  $q$  items, then the most that can be sold is  $\min(D, q)$ . That is, if  $D$  is bigger than  $q$ , you can only sell  $q$ . If  $D$  is smaller than  $q$ , you can only sell  $D$ .

What is the possible amount left over (available for salvage)? If  $D$  is the demand and we start with  $q$ , then there are two cases: (i) demand is less than the amount ordered ( $D < q$ ) or (ii) demand is greater than or equal to the amount ordered ( $D \geq q$ ). In case 1, we will have  $q - D$  items left over. In case 2, we will have 0 left over. Thus, the amount left over at the end of the day is  $\max(0, q - D)$ . Since the news vendor buys  $q$  items for  $c$ , the cost of the items for the day is  $c \times q$ . Thus, the profit has the following form:

$$G(D, q) = s \times \min(D, q) + u \times \max(0, q - D) - c \times q \quad (3.6)$$

In words, the profit is equal to the sales revenue plus the salvage revenue minus the ordering cost. The sales revenue is the selling price times the amount sold. The salvage revenue is the salvage value times the amount left over. The ordering cost is the cost of the items times the amount ordered. To find the optimal value of  $q$ , we should try to maximize the expected profit. Since  $D$  is a random variable,  $G(D, q)$  is also a random variable. Taking the expected value of both sides of Equation (3.6) yields

$$g(q) = E[G(D, q)] = sE[\min(D, q)] + uE[\max(0, q - D)] - cq$$

Whether or not this equation can be optimized depends on the form of the distribution of  $D$ ; however, simulation can be used to estimate this expected value for any given  $q$ . Let us look at an example.

### EXAMPLE 3.2 Sly's BBQ Wing Problem

Sly's convenience store sells a piece of BBQ wings for 25 cents. They cost 15 cents to make a piece. The BBQ wings that are not sold on a given day are purchased by a local food pantry for 2 cents each. Assuming that Sly decides to make 30 wings a day, what is the expected revenue for the wings, provided that the demand distribution is as shown in Table 3.2.

**TABLE 3.2 Distribution of BBQ Wing Demand**

$d_i$	5	10	40	45	50	55	60
$f(d_i)$	0.1	0.2	0.3	0.2	0.1	0.05	0.05
$F(d_i)$	0.1	0.3	0.6	0.8	0.9	0.95	1.0

To get started with this spreadsheet model, first, conceptualize how you want to layout the spreadsheet. A good place to start is to have well-labeled inputs at the top of the spreadsheet. Figure 3.8 illustrates this arrangement. Notice how the inputs and decision variables are clearly labeled and the demand distribution is at the top of the spreadsheet. The simulation outputs are labeled in the middle of the spreadsheet. Finally, the data table used to perform the simulation is at the bottom. Locating the data table at the bottom facilitates

	A	B	C	D	E	F	G
1	<b>Sly's Convenience Store Example</b>						
2	<b>Inputs</b>						
3	cost per unit c =	0.15					
4	price per unit s =	0.25		0.1	0	0.1	5
5	salvage per unit u =	0.02		0.2	0.1	0.3	10
6				0.3	0.3	0.6	40
7	<b>Decision Variable</b>						
8	Order Quantiy q =	30		0.2	0.6	0.8	45
9				0.1	0.8	0.9	50
10	<b>Outputs</b>						
11	Daily Demand	5		0.05	0.95	1	60
12	Amount Sold	5		u =	0		
13	Amount Left Over	25		D =	5		
14	Sales Revenue	1.25					
15	Salvage Revenue	0.5					
16	Ordering Cost	4.5					
17	Profit	-2.75					
18	<b>Simulation Summary</b>						
19	Average =	1.528					
20	<b>Data Table</b>						
21		-2.75					
22		1	3				
23		2	3				
24		3	-1.6				
25		4	-1.6				
26		95	3				
27		96	-1.6				
118		97	-1.6				
119		98	-2.75				
120		99	-1.6				
121		100	3				
122							
123							

**Figure 3.8** Sly's convenience store spreadsheet model.

adding additional rows to the data table. Also note that there are well-labeled intermediate calculations. Not only is the profit visible but also the components of the profit (e.g., sales revenue and salvage revenue) are also visible. This facilitates further analysis of the components of the simulation and also debugs the simulation. Figure 3.9 illustrates the spreadsheet formulas for this model. Note that a look-up table is used to generate the demands and that a data table is used to simulate the daily profit.

### 3.3.3 Simulating a Random Cash Flow

This section looks at a somewhat more complex situation involving an analysis of a cash flow that has random elements. As you may recall, cash flow analysis is commonly performed to evaluate different alternatives from an engineering economic standpoint. Within an introductory presentation of engineering economics, the cash flows do not have any random components. However, in practice, it is often very difficult to get exact values for cash flow analysis. The elements are often estimated from data or developed through subjective estimates. A simulation approach to the problem allows for the inherent randomness to be taken into account and provides a way to assess the risk associated with the decisions that are made based on the cash flow analysis. Example 3.3 illustrates these concepts.

#### EXAMPLE 3.3 Random Cash Flow Analysis

A firm is trying to decide whether or not it should purchase a new scale to check a package filling line in the plant. The scale would allow for better control over the filling

A	B	C	D	E	F	G
1 Sly's Convenience S						
2 Inputs						
3 cost per unit c = 0.15			Demand PMF			
4 price per unit s = 0.25		0.1	0		=D4	5
5 salvage per unit u = 0.02		0.2	=F4		=F4+D5	10
6		0.3	=F5		=F5+D6	40
7 Decision Variable		0.2	=F6		=F6+D7	45
8 Order Quanty q = 30		0.1	=F7		=F7+D8	50
9		0.05	=F8		=F8+D9	55
10 Outputs		0.05	=F9		=F9+D10	60
11 Daily Demand	=E13					
12 Amount Sold	=MIN(B11,B8)					
13 Amount Left Over	=MAX(0,B8-B11)			u = =RAND()		
14 Sales Revenue	=B12*B4			D = =VLOOKUP(E12,E4:G10,3)		
15 Salvage Revenue	=B13*B5					
16 Ordering Cost	=B8*B3					
17 Profit	=B14+B15-B16					
18						
19 Simulation Summary						
20 Average =	=AVERAGE(B24:B123)					
21						
22 Data Table						
23	=B17					
24 1	=TABLE(A23)					
25 2	=TABLE(A23)					
26 3	=TABLE(A23)					
27 4	=TABLE(A23)					

**Figure 3.9** Spreadsheet formulas for Sly's convenience store spreadsheet model.

operation and result in less overfilling. It is known for certain that the scale costs \$800 initially. The annual cost has been estimated to be normally distributed with a mean of \$100 and a standard deviation of \$10. The extra savings associated with better control of the filling process has been estimated to be normally distributed with a mean of \$300 and a standard deviation of \$50. The salvage value has been estimated to be uniformly distributed between \$90 and \$100. The useful life of the scale varies according to the amount of usage of the scale. The manufacturing has estimated that the useful life can vary between 4 and 7 years with the chances given in Table 3.3.

The interest rate has been varying recently, and the firm is unsure of the rate for performing the analysis. To be safe, they have decided that the interest rate should be modeled as a beta random variable over the range from 6 to 9 with alpha = 5.0 and beta = 1.5. Given all the uncertain elements in the situation, they have decided to perform a simulation analysis in order to assess the expected present value of the decision and the chance that the decision has a negative return.

In order to develop a solution to Example 3.3, a basic understanding of the formulas available within Excel™ involving cash flows is required. This problem is an economic analysis problem and can be solved using present value calculations. The Excel™ function PV will compute the present value of a future value that occurs at the end of a period as follows:

$$P = F \left[ \frac{1}{(1 + i)^n} \right] = F(P/F, i, n) = PV(i, n, \cdot, -F) \quad (3.7)$$

**TABLE 3.3** Useful Life Distribution in Years

Years	4	5	6	7
f(years)	0.3	0.4	0.1	0.2
F(years)	0.3	0.7	0.8	1.0

Notice that PV has optional parameters that determine its use. The function PV will compute the present value of a series of future values that are assumed to occur at the end of the periods as follows:

$$P = A \left[ \frac{(1+i)^n - 1}{i(1+i)^n} \right] = A(P/A, i, n) = PV(i, n, -A) \quad (3.8)$$

However, in computing the present value of the salvage value, annual cost, and annual savings, you should see that the inputs to the above functions will be random variables in this problem. That is,  $i$ ,  $n$ ,  $F$ , and  $A$  must be determined from the given probability distributions. The basic formulation is as follows:

1. Let  $S \sim U(90, 100)$  be a random variable that represents the salvage value.
2. Let  $C \sim N(\mu = 100, \sigma = 10)$  be a random variable that represents the annual cost.
3. Let  $Y \sim N(\mu = 300, \sigma = 50)$  be a random variable that represents the annual savings.
4. Let  $N$  be a random variable that represents the useful life with distribution give in Table 3.3.
5. Let  $I$  be a random variable that represents the interest rate.
6. Present value of cost:  $P_1 = C(P/A, I, N)$ .
7. Present value of savings:  $P_2 = Y(P/A, I, N)$ .
8. Present value of salvage value:  $P_3 = S(P/F, I, N)$ .
9. Net Present Value:  $P = -800 - P_1 + P_2 + P_3$ .

Using the PV() function and randomly generated values for the salvage value, annual cost, annual savings, useful life, and the interest rate, you can simulate values of the net present value.

The following discusses how to set up the spreadsheet to accomplish the solution to Example 3.3. A good spreadsheet simulation should be separated into three parts: 1) the inputs, 2) the outputs, and 3) the simulation. The inputs and outputs will be discussed in this section. Section 3.4 will discuss the simulation data table implementation. Figure 3.10 shows the input area for the simulation. The distribution's parameters have been organized into a common area. This allows for easily changing the parameters to rerun different simulations.

Figure 3.11 shows the spreadsheet formulas used to implement the simulation. In this example, the simulation is tied to the input area and uses the NORMINV function to simulate the annual cost and annual savings. The salvage value is simulated by cell B27, where a  $U(0, 1)$  is shifted and scaled over the desired range. This provides  $S \sim U(90, 100)$  via the inverse transform technique. The useful life is simulated using the VLOOKUP function. Finally, the interest rate is simulated using the BETAINV function. Then the PV() function is used to compute the desired present values.

As can be seen in Figure 3.12, the equations provide one observation of the present value. Because of randomness, it would be foolish to base our decision concerning this cash flow opportunity solely on one observation. The next section discusses how many observations we need in order to be confident in our decision.

	A	B	C	D	E
1	Inputs				
2	Initial Cost	-\$800.00			
3	Annual Cost				
4	mean	-\$100.00			
5	std dev	\$10.00			
6	Annual Savings				
7	mean	\$300.00			
8	std dev	\$50.00			
9	Salvage Value				
10	min	\$90.00			
11	max	\$100.00			
12	Useful Life	PMF	LR	CDF	Life
13		0.30	0	0.3	4
14		0.40	0.3	0.7	5
15		0.10	0.7	0.8	6
16		0.20	0.8	1	7
17	Interest Rate				
18	alpha	5.00			
19	beta	1.50			
20	min	6.00%			
21	max	9.00%			

**Figure 3.10** Random cash flow simulation inputs.

	A	B	C	
22				
23	Outputs			
24	Initial Cost	-\$800.00		
25	Annual Cost	-102.069		
26	Annual Savings	183.9268		
27	Salvage Value	\$98.14		
28	Useful Life	7		
29	Interest Rate	0.072705		
30	PV of Salvage	\$60.05		
31	PV of Annual Cost	(\$544.93)		
32	PV of Annual Savings	\$981.96		
33	Net Present Value	-\$302.93		

**Figure 3.11** Random cash flow simulation output results.

### 3.4 INTRODUCTORY STATISTICAL CONCEPTS

The spreadsheet simulation models that have been illustrated have estimated the quantity of interest with a single point estimate (i.e.,  $\bar{X}$ ). For example, in Figure 3.7, we know that the true area under the curve is 4.66; however, the point estimate returned by the AVERAGE() function was a value of 4.74, based on 20 samples. Thus, there is sampling error in our estimate. The key question examined in this section is how to control the sampling error in a simulation experiment. The approach that we will take is to determine the number of samples so that we can have high confidence in our point estimate. In order to address these issues, we need to review some basic statistical concepts.

#### 3.4.1 Point Estimates and Confidence Intervals

Let  $x_i$  represent the  $i^{th}$  observations in a sample,  $x_1, x_2, \dots, x_n$ , of size  $n$ . Represent the random variables in the sample as  $X_1, X_2, \dots, X_n$ . The random variables form a random

	A	B
22		
23	<b>Outputs</b>	
24	Initial Cost	=B2
25	Annual Cost	=NORMINV(RAND(),B4,B5)
26	Annual Savings	=NORMINV(RAND(),B7,B8)
27	Salvage Value	=B10+(B11-B10)*RAND()
28	Useful Life	=VLOOKUP(RAND(),C13:E16,3)
29	Interest Rate	=BETAINV(RAND(),B18,B19,B20,B21)
30	PV of Salvage	=PV(B29,B28,-B27)
31	PV of Annual Cost	=PV(B29,B28,-B25)
32	PV of Annual Savings	=PV(B29,B28,-B26)
33	Net Present Value	=B24+B31+B32+B30

**Figure 3.12** Random cash flow simulation output formulas.

sample, if 1) the  $X_i$  are independent random variables and 2) every  $X_i$  has the same probability distribution. Let us assume that these two assumptions have been established. Denote the unknown CDF of  $X$  as  $F(x)$  and define the unknown expected value and variance of  $X$  with  $E[X] = \mu$  and  $Var[X] = \sigma^2$ , respectively.

A statistic is any function of the random variables in a sample. Any statistic that is used to estimate an unknown quantity based on the sample is called an estimator. What would be a good estimator for the quantity  $E[X] = \mu$ ? Without going into the details of the meaning of statistical goodness, one should remember that the sample average is generally a good estimator for  $E[X] = \mu$ . Define the sample average as follows:

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i \quad (3.9)$$

Notice that  $\bar{X}$  is a function of the random variables in the sample, and therefore, it is also a random variable. This makes  $\bar{X}$  a statistic, since it is a function of the random variables in the sample.

Any random variable has a corresponding probability distribution. The probability distribution associated with a statistic is called its sampling distribution. The sampling distribution of a statistic can be used to form a confidence interval on the point estimate associated with the statistic. The point estimate is simply the value obtained from the statistic once the data has been realized. The point estimate for the sample average is computed from the sample:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (3.10)$$

A confidence interval expresses a degree of certainty associated with a point estimate. A specific confidence interval does not imply that the parameter  $\mu$  is inside the interval. In fact, the true parameter is either in the interval or not within the interval. Instead you should think about the confidence level  $1 - \alpha$  as an assurance about the procedure used to compute the interval. That is, a confidence interval procedure ensures that if a large number of confidence intervals are computed each based on  $n$  samples, then the proportion of the confidence intervals that actually contain the true value of  $\mu$  should be close to  $1 - \alpha$ . The value  $\alpha$  represents risk that the confidence interval procedure will produce a specific interval that does not contain the true parameter value. Any one particular confidence interval will

either contain the true parameter of interest or it will not. Since you do not know the true value, you can use the confidence interval to assess the risk of making a bad decision based on the point estimate. You can be confident in your decision making or conversely know that you are taking a risk of  $\alpha$  of making a bad decision. Think of  $\alpha$  as the risk that using the confidence interval procedure will get you fired. If we know the sampling distribution of the statistic, then we can form a confidence interval procedure.

Under the assumption that the sample size is large enough such that the distribution of  $\bar{X}$  is normally distributed, you can form an approximate confidence interval on the point estimate,  $\bar{x}$ . Assuming that the sample variance:

$$s^2(n) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \quad (3.11)$$

is a good estimator for  $\text{Var}[X] = \sigma^2$ , then a  $(1 - \alpha)100\%$  confidence interval estimate for  $\mu$  is:

$$\bar{x} \pm t_{\alpha/2,n-1} \frac{s}{\sqrt{n}} \quad (3.12)$$

where  $t_{\alpha/2,n-1}$  is the upper  $100(1 - \alpha/2)$  percentage point of the Student  $t$ -distribution with  $n - 1$  degrees of freedom. The Excel™ function T.INV( $p$ , degrees freedom) function computes the *left-tailed* Student  $t$ -distribution value for a given probability and degrees of freedom. Thus, in order to get the right-tailed  $t$ -value, you need to subtract 1 from the probability. The spreadsheet tab labeled *TINV* in the spreadsheet that accompanies this chapter illustrates this function. Thus, in order to compute  $t_{\alpha/2,n-1}$ , use the following formula:

$$t_{\alpha/2,n-1} = \text{T.INV}(1 - \alpha/2, n - 1) \quad (3.13)$$

### 3.4.2 Determining the Sample Size

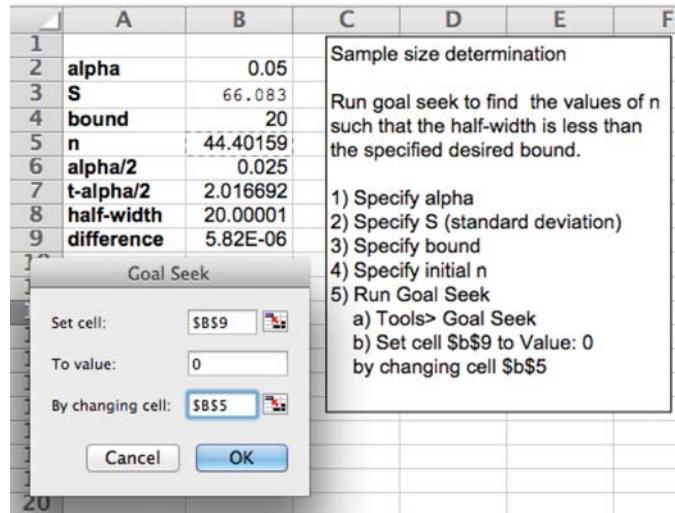
The confidence interval for a point estimator can serve as the basis for determining how many observations to have in the sample. From Equation (3.12), the quantity

$$h = t_{\alpha/2,n-1} \frac{s}{\sqrt{n}} \quad (3.14)$$

is called the half-width of the confidence interval. You can place a bound,  $E$ , on the half-width by picking a sample size that satisfies:

$$h = t_{\alpha/2,n-1} \frac{s}{\sqrt{n}} \leq E \quad (3.15)$$

Unfortunately,  $t_{\alpha/2,n-1}$  depends on  $n$ , and thus Equation (3.15) is an iterative equation. That is, you must try different values of  $n$  until the condition is satisfied. Figure 3.13 is a snapshot from the spreadsheet tab labeled *SampleSize* that accompanies this chapter's spreadsheet. In the spreadsheet, goal seek is used to iteratively search for the required value of  $n$ .



**Figure 3.13** Using goal seek to iteratively determine the sample size.

Alternatively, the required sample size can be approximated using the normal distribution. Solving Equation (3.15) for  $n$  yields

$$n \geq \left( \frac{t_{\alpha/2,n-1} s}{E} \right)^2 \quad (3.16)$$

As  $n$  gets large,  $t_{\alpha/2,n-1}$  converges to the upper  $100(1 - \alpha/2)$  percentage point of the standard normal distribution  $z_{\alpha/2}$ . This yields the following approximation:

$$n \geq \left( \frac{z_{\alpha/2} s}{E} \right)^2 \quad (3.17)$$

This equation generally works well for large  $n$ , say  $n > 50$ . Both of these methods require an initial value for the standard deviation. In order to use these methods, you should make an initial pilot sample (e.g.,  $n = 5$ ) in order to get an initial estimate of the standard deviation. Given a value for  $s$ , you can then set a desired bound and use the formulas. The bound is problem and performance measure dependent and is under your subjective control. You must determine what bound is reasonable for your given situation. One thing to remember is that the bound is squared in the denominator for evaluating  $n$ . Thus, very small values of  $E$  can result in very large sample sizes.

#### EXAMPLE 3.4 Determining the Sample Size

Suppose we are required to estimate the output from a spreadsheet simulation so that we are 99% confidence that we are within  $\pm 0.1$  of the true population mean. After taking a pilot sample of size  $n = 10$ , we have estimated  $s = 6$ . Recommend a suitable sample size.

We will use Equation (3.17). For a 99% confidence interval, we have  $\alpha = 0.01$  and  $\alpha/2 = 0.005$ . Thus,  $z_{0.005} = 2.576$ . This yields

$$n \geq \left( \frac{z_{\alpha/2} s}{E} \right)^2 = \left( \frac{2.576 \times 6}{0.1} \right)^2 = 23,888.8 = 23,889$$

If the quantity of interest is a proportion, then a different method can be used. In particular, a  $100 \times (1 - \alpha)\%$  large sample confidence interval for a proportion,  $p$ , has the following form:

$$\hat{p} \pm z_{\alpha/2} \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}} \quad (3.18)$$

where  $\hat{p}$  is the estimate for  $p$ . From this, you can determine the sample size via the following equation:

$$n = \left( \frac{z_{\alpha/2}}{E} \right)^2 \hat{p}(1 - \hat{p}) \quad (3.19)$$

Again, a pilot run is necessary for obtaining an initial estimate,  $\hat{p}$ , for use in determining the sample size. If no pilot run is available, then  $\hat{p} = 0.5$  is often assumed.

If you have more than one performance measure of interest, you can use these sample size techniques for each of your performance measures and then use the maximum sample size required across the performance measures. Let us illustrate these methods on Example 3.3.

### EXAMPLE 3.5 Random Cash Flow Analysis (continued)

Suppose we are interested in estimating the expected present value of the situation described in Example 3.3. We desire to be 95% confident that our estimate of the true expected present value is within  $\pm \$10$ . In addition, we are interested in estimating the probability that the present value is negative. That is, we want to estimate the risk of losing money on the situation.

To solve Example 3.5, the spreadsheet needs to be modified to allow a small pilot to be run and also estimate the probability that the present value is negative. To estimate a probability, it is useful to define an indicator variable. An indicator variable has the value 1 if the condition associated with the probability is true and has the value 0 if it is false.

Let  $X$  represent the unknown random present value. Then, we are interested in estimating  $E[X]$  and  $p = P\{X < 0\}$ . To estimate these quantities, we generate a random sample,  $X_1, X_2, \dots, X_n$ .  $E[X]$  can be estimated using  $\bar{X}$ . To estimate  $p$ , define the following indicator variable:

$$Y_i = \begin{cases} 1 & X_i < 0 \\ 0 & X_i \geq 0 \end{cases} \quad (3.20)$$

This definition of the indicator variable  $Y_i$  allows  $\bar{Y}$  to be used to estimate  $p$ . Recall the definition of the sample average  $\bar{Y}$ . Since  $Y_i$  is a 1 only if  $X_i < 0$ , then the  $\sum_{i=1}^n Y_i$  simply adds up the number of 1's in the sample. Thus,  $\sum_{i=1}^n Y_i$  represents the count of the number of times the event  $X_i < 0$  occurred in the sample. Call this  $\#\{X_i < 0\}$ . The ratio of the

number of times an event occurred to the total number of possible occurrences represents the proportion. Thus, an estimator for  $p$  is

$$\hat{p} = \bar{Y} = \frac{1}{n} \sum_{i=1}^n Y_i = \frac{\#\{X_i < 0\}}{n}$$

Therefore, computing the average of an indicator variable will estimate the desired probability. To implement the indicator variable in the spreadsheet, the following spreadsheet formula can be used: IF(cell reference < 0, 1, 0), where cell reference refers to an observation of the present value.

Figure 3.14 illustrates the spreadsheet setup for simulating multiple cash flows using a data table. Notice how the spreadsheet statistical functions are utilized to summarize the results of the simulation. The formulas AVERAGE(), STDEV(), COUNT(), and T.INV() are used to implement Equation (3.12). These formulas are active formulas summarizing the results of the simulation. If the spreadsheet is recalculated, then the summary statistics will be updated. Figure 3.15 illustrates the results based on a pilot of  $n = 30$  observations. Notice that the average present value is 85.16 and the standard deviation is quite large at 223. This results in a very wide confidence interval of (1.88, 168.43). To meet the specification of a

	G	H	I
1	<b>Simulation Statistics</b>		
2	<b>Sample Average</b>	=AVERAGE(H20:H49)	=AVERAGE(I20:I49)
3	<b>Standard Error</b>	=H5/SQRT(H11)	=I5/SQRT(I11)
4	<b>Median</b>	=MEDIAN(H20:H49)	=MEDIAN(I20:I49)
5	<b>Standard Deviation</b>	=STDEV(H20:H49)	=STDEV(I20:I49)
6	<b>Sample Variance</b>	=VAR(H20:H49)	=VAR(I20:I49)
7	<b>Range</b>	=H9-H8	=I9-I8
8	<b>Minimum</b>	=MIN(H20:H49)	=MIN(I20:I49)
9	<b>Maximum</b>	=MAX(H20:H49)	=MAX(I20:I49)
10	<b>Sum</b>	=SUM(H20:H49)	=SUM(I20:I49)
11	<b>Count</b>	=COUNT(H20:H49)	=COUNT(I20:I49)
12	<b>Alpha</b>	0.05	0.05
13	<b>T-Value(Alpha/2,n-1)</b>	=T.INV(1-H12/2,H11-1)	=T.INV(1-I12/2,I11-1)
14	<b>CI Half-Width</b>	=H13*H3	=I13*I3
15	<b>LL-Confidence Interval</b>	=H2-H14	=I2-I14
16	<b>UL-Confidence Interval</b>	=H2+H14	=I2+I14
17			
18	<b>Simulation Data Table</b>		
19		=B33	
20	1	=TABLE(G19)	=IF(H20<0,1,0)
21	2	=TABLE(G19)	=IF(H21<0,1,0)
22	3	=TABLE(G19)	=IF(H22<0,1,0)
23	4	=TABLE(G19)	=IF(H23<0,1,0)
24	5	=TABLE(G19)	=IF(H24<0,1,0)
25	6	=TABLE(G19)	=IF(H25<0,1,0)
26	7	=TABLE(G19)	=IF(H26<0,1,0)
27	8	=TABLE(G19)	=IF(H27<0,1,0)
28	9	=TABLE(G19)	=IF(H28<0,1,0)
29	10	=TABLE(G19)	=IF(H29<0,1,0)

**Figure 3.14** Cash flow simulation summary formulas.

	G	H	I	J	
1	<b>Simulation Statistics</b>	<b>NPV</b>	<b>P(NPV &lt; 0)</b>		
2	<b>Sample Average</b>	85.16036	0.4		
3	<b>Standard Error</b>	40.71579	0.090972		
4	<b>Median</b>	61.3453	0		
5	<b>Standard Deviation</b>	223.0095	0.498273		
6	<b>Sample Variance</b>	49733.26	0.248276		
7	<b>Range</b>	891.265	1		
8	<b>Minimum</b>	-387.753	0		
9	<b>Maximum</b>	503.5121	1		
10	<b>Sum</b>	2554.811	12		
11	<b>Count</b>	30	30		
12	<b>Alpha</b>	0.05	0.05		
13	<b>T-Value(Alpha/2,n-1)</b>	2.04523	2.04523		
14	<b>CI Half-Width</b>	83.27313	0.186058		
15	<b>LL-Confidence Interval</b>	1.887226	0.213942		
16	<b>UL-Confidence Interval</b>	168.4335	0.586058		
17					
18	<b>Simulation Data Table</b>				
19		-\$344.64			
20		1 423.6894	0 =IF(H20<0,1,0)		
21		2 -247.635	1		
22		3 -167.293	1		
23		4 83.08767	0		
24		5 439.5662	0		
25		6 245.4963	0		
26		7 163.4001	0		
27		8 -55.7688	1		
28		9 -97.0344	1		
29		10 -207.699	1		

Figure 3.15 Cash flow simulation results for 30 observations.

half-width of \$10 will require a larger sample. Using Equation (3.17), we get the following:

$$\begin{aligned} n &\geq \left(\frac{z_{\alpha/2} s}{E}\right)^2 \\ &\geq \left(\frac{1.96 \times 223}{10}\right)^2 \approx 1910.3 \end{aligned}$$

Thus, choosing  $n = 1911$  should allow us to meet the half-width criteria. Figure 3.16 illustrates the results for  $n = 1911$ . Notice that the estimated half-width is very close to the desired \$10 at \$10.88. The results for  $n = 1911$  indicate that we can be 95% confident that the true expected present value for the situation is between (58.243, 79.997). However, because of the large amount of variability in the cash flow, the probability of having a negative present value is considerable, estimated to be very close to 42%. This would appear to be a relatively risky situation.

A question that might arise is why the resulting half-width value was slightly over the desired value. This suggests that slightly more than  $n = 1911$  is needed to get under \$10 for the half-width. Equation (3.17) is only an approximation, and it is based on a pilot sample of 30 observations. Notice that the sample standard deviation value (242.0) for  $n = 1911$  is larger than the value (223.0) when  $n = 30$ . Thus, there was considerable sampling error associated with the pilot sample; however, it still gave a very reasonable suggestion for the sample size.

Based on these examples, you should have a basic understanding of how simulation can be performed within a spreadsheet. You have also learned how to generate random

	G	H	I
1	<b>Simulation Statistics</b>	<b>NPV</b>	<b>P(NPV &lt; 0)</b>
2	<b>Sample Average</b>	69.12068	0.425955
3	<b>Standard Error</b>	5.546058	0.011315
4	<b>Median</b>	49.18056	0
5	<b>Standard Deviation</b>	242.4458	0.494616
6	<b>Sample Variance</b>	58779.99	0.244645
7	<b>Range</b>	1668.144	1
8	<b>Minimum</b>	-713.756	0
9	<b>Maximum</b>	954.3876	1
10	<b>Sum</b>	132089.6	814
11	<b>Count</b>	1911	1911
12	<b>Alpha</b>	0.05	1.05
13	<b>T-Value(Alpha/2,n-1)</b>	1.961207	-0.06272
14	<b>CI Half-Width</b>	10.87697	-0.00071
15	<b>LL-Confidence Interval</b>	58.24371	0.426665
16	<b>UL-Confidence Interval</b>	79.99765	0.425245

**Figure 3.16** Cash flow simulation results for sample size  $n = 1911$ .

variables from a variety of probability distributions within a spreadsheet. These concepts should give you a better understanding of how to treat randomness in simulation. There are a number of sophisticated software packages that facilitate spreadsheet simulation that you should explore if you are interested in furthering your understanding of simulation within spreadsheets. While Excel™ formed the basis of these examples, the methodologies should translate very well to other spreadsheet software.

### 3.5 SUMMARY

This chapter covered a number of important concepts used within spreadsheet simulation including the following:

- Generating random numbers and random variables
- Introduction to Monte-Carlo methods
- How to set up a spreadsheet to perform simulation
- Introductory statistical concepts in simulation.

The spreadsheet modeling performed in this chapter illustrated how a simulation involving random components is really a statistical experiment. The output from the simulation must be analyzed using statistical models and the experiments must be planned carefully in order to control sampling error.

However, a spreadsheet environment is quite limiting in terms of modeling systems that have complex dynamic components that evolve over time. The next chapter introduces you to modeling concepts associated with discrete-event dynamic simulation.

### EXERCISES

- 3.1 Consider the multiplicative congruential generator with  $a = 13$ ,  $m = 64$ , and seeds  $X_0 = 1,2,3,4$ . Set up a spreadsheet to generate from this generator.

- 3.2 Consider the multiplicative congruential generator with  $a = 11$ ,  $m = 64$ , and seeds  $X_0 = 1, 2, 3, 4$ . Set up a spreadsheet to generate from this generator.
- 3.3 Generate 1000 uniform  $(0, 1)$  numbers using a spreadsheet.
- Use the Kolmogorov–Smirnov test with  $\alpha = 0.05$  to test if the hypothesis that the numbers are uniformly distributed on the  $(0, 1)$  interval can be rejected.
  - Use a chi-square goodness-of-fit test with 10 intervals and  $\alpha = 0.05$  to test if the hypothesis that the numbers are uniformly distributed on the  $(0, 1)$  interval can be rejected.
  - Test the hypothesis that the numbers are uniformly distributed within the unit square,  $\{(x, y) : x \in (0, 1), y \in (0, 1)\}$  using the 2D chi-squared test at a 95% confidence level. Use 10 intervals for each of the dimensions.
  - Test the hypothesis that the numbers have a lag-1 correlation of zero. Make an autocorrelation plot of the numbers.
  - If you have access to a suitable statistical package, perform a runs above the mean test to check the randomness of the numbers. Use the nonparametric runs test functionality of your statistical software to perform this test.
  - What is your conclusion concerning the suitability of your spreadsheet's random number generator?
- 3.4 Write a one line spreadsheet formula to generate Bernoulli random variables with success probability, 0.35.
- 3.5 Write a one line spreadsheet formula to generate random variables from a normal distribution with mean 10.0 and variance 4.0.
- 3.6 Write a one line spreadsheet formula to generate random variables from an exponential distribution with a rate parameter of 5 per hour.
- 3.7 The service times for an automated storage and retrieval system has a shifted exponential distribution. It is known that it takes a minimum of 15 seconds for any retrieval. The parameter of the exponential distribution is  $\lambda = 45$ . Set up a spreadsheet that will generate 20 observations of the service times.
- 3.8 The time to failure for a computer printer fan has the Weibull distribution with shape parameter  $\alpha = 2$  and scale parameter  $\beta = 3$ . Set up a spreadsheet that will generate 10 failure times for the computer printer fan.
- 3.9 The time to failure for a computer printer fan has the Weibull distribution with shape parameter  $\alpha = 2$  and scale parameter  $\beta = 3$ . Testing has indicated that the distribution is limited to the range from 1.5 to 4.5.
- Set up a spreadsheet to generate 100 observations from this truncated distribution.
  - Using your favorite software, make a histogram of your observations.
- 3.10 The interest rate for a capital project is unknown. An accountant has estimated that the minimum interest rate will be between 2% and 5% within the next year. The accountant believes that any interest rate in this range is equally likely. You are tasked with generating interest rates for a cash flow analysis of the project. Set up a spreadsheet that will generate five interest rate values for the capital project analysis.

- 3.11 Set up a spreadsheet to generate 30 observations from the following probability density function:

$$f(x) = \begin{cases} \frac{3x^2}{2} & -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

- 3.12 Suppose that the service time for a patient consists of two distributions. There is a 25% chance that the service time is uniformly distributed with minimum of 20 minutes and a maximum of 25 minutes, and a 75% chance that the time is distributed according to the Weibull distribution with shape of 2 and a scale of 4.5.

- (a) Set up a spreadsheet to generate 100 observations of the service time.
- (b) Using your favorite software, make a histogram of your observations.
- (c) Compute the theoretical expected value of the distribution.
- (d) Estimate the expected value of the distribution and compute a 95% confidence interval on the expected value. Did your confidence interval contain the theoretical expected value of the distribution?

- 3.13 Suppose that  $X$  is a random variable with a  $N(\mu = 2, \sigma = 1.5)$  normal distribution. Generate 100 observations of  $X$  using a spreadsheet.

- (a) Estimate the mean from your observations. Report a 95% confidence interval for your point estimate.
- (b) Estimate the variance from your observations. Report a 95% confidence interval for your point estimate.
- (c) Estimate the  $P\{X \geq 3\}$  from your observations. Report a 95% confidence interval for your point estimate.

- 3.14 Samples of 20 parts from a metal grinding process are selected every hour. Typically, 2% of the parts needs rework. Let  $X$  denote the number of parts in the sample of 20 that require rework. A process problem is suspected if  $X$  exceeds its mean by more than 3 standard deviations. Using a spreadsheet, simulate 30 hours of the process, that is, 30 samples of size 20, and estimate the chance that  $X$  exceeds its expected value by more than 3 standard deviations.

- 3.15 Consider the following discrete distribution of the random variable  $X$  whose PMF is  $p(x)$ .

$x$	0	1	2	3	4
$p(x)$	0.3	0.2	0.2	0.1	0.2

- (a) Create a look-up table that can be used to determine a sample from the discrete distribution,  $p(x)$ . See Example 2.9.
- (b) Generate 30 observations of the random variable  $X$  using your spreadsheet.

- 3.16 Suppose that customers arrive at an ATM via a Poisson process with mean 7 per hour. Develop a spreadsheet to generate the arrival times of the first six customers.

- 3.17 The demand for parts at a repair bench per day can be described by the following discrete PMF:

Demand	0	1	2
Probability	0.3	0.2	0.5

Generate the demand for the first 4 days using a spreadsheet implementation.

- 3.18 Customers arrive at a service location according to a Poisson distribution with mean 10 per hour. The installation has two servers. Experience shows that 60% of the arriving customers prefer the first server. Set up a spreadsheet that will determine the arrival times of the first three customers at each server.
- 3.19 Suppose  $n = 10$  observations were collected on the time spent in a manufacturing system for a part. The analysis determined a 95% confidence interval for the mean system time of [18.595, 32.421].
- Find the approximate number of samples needed to have a 95% confidence interval that is within  $\pm 2$  minutes of the true mean system time.
  - Find the approximate number of samples needed to have a 99% confidence interval that is within  $\pm 1$  minute of the true mean system time.
- 3.20 Using a spreadsheet and the Monte-Carlo method, estimate the following integral with 95% confidence to within  $\pm 0.01$ .

$$\int_1^4 \left( \sqrt{x} + \frac{1}{2\sqrt{x}} \right) dx$$

- 3.21 Using a spreadsheet and the Monte-Carlo method, estimate the following integral with 99% confidence to within  $\pm 0.01$ .

$$\int_0^\pi (\sin(x) - 8x^2) dx$$

- 3.22 Using a spreadsheet and the Monte-Carlo method, estimate the following integral with 99% confidence to within  $\pm 0.01$ .

$$\theta = \int_0^1 \int_0^1 (4x^2y + y^2) dx dy$$

- 3.23 Consider the triangular distribution:

$$F(x) = \begin{cases} 0 & x < a \\ \frac{(x-a)^2}{(b-a)(c-a)} & a \leq x \leq c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)} & c < x \leq b \\ 1 & b < x \end{cases}$$

- (a) Implement a VBA function with the following signature to compute the inverse CDF of a triangular distribution with parameters min =  $a$ , mode =  $c$ , and max =  $b$ . Also,  $u$  is a number between 0 and 1.

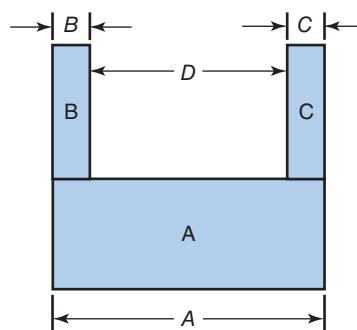
```
Public Function invTriangularCDF(min As Double, mode As
Double,
max As Double, double u) As Double
End Function
```

- (b) Use a spreadsheet to generate 1000 observations of the triangular distribution with  $a = 2$ ,  $c = 5$ , and  $b = 10$ .
- (c) Use your favorite statistical software to make a histogram of 1000 observations from your implementation of the triangular distribution with  $a = 2$ ,  $c = 5$ , and  $b = 10$ .
- 3.24 A firm is trying to decide whether or not to invest in two proposals A and B that have the net cash flows shown in the following table, where  $N(\mu, \sigma)$  represents that the cash flow value comes from a normal distribution with the provided mean and standard deviation.

End of Year	0	1	2	3	4
A	$N(-250, 10)$	$N(75, 10)$	$N(75, 10)$	$N(175, 20)$	$N(150, 40)$
B	$N(-250, 5)$	$N(150, 10)$	$N(150, 10)$	$N(75, 20)$	$N(75, 30)$

The interest rate has been varying recently and the firm is unsure of the rate for performing the analysis. To be safe, they have decided that the interest rate should be modeled as a beta random variable over the range from 2% to 7% with  $\alpha_1 = 4.0$  and  $\alpha_2 = 1.2$ . Given all the uncertain elements in the situation, they have decided to perform a simulation analysis in order to assess the situation.

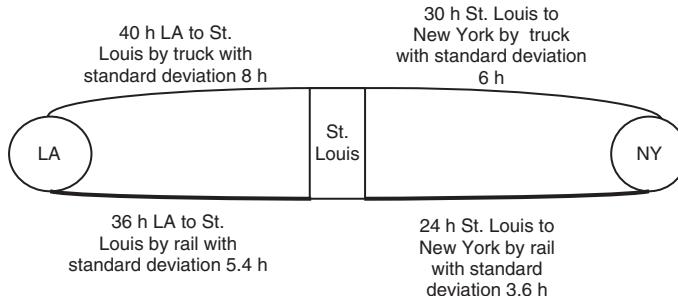
- (a) Compare the expected present worth of the two alternatives. Estimate the probability that alternative A has a higher present worth than alternative B.
- (b) Determine the number of samples needed to be 95% confidence that you have estimated the  $P\{\bar{PV}(A) > \bar{PV}(B)\}$  to within  $\pm 0.10$ , where  $PV(A)$  and  $PV(B)$  is the present value of alternatives A and B, respectively.
- 3.25 A U-shaped component is to be formed from the three parts A, B, and C. The picture is shown in the figure below. The length of A is lognormally distributed with a mean of 20 millimeters and a standard deviation of 0.2 millimeter. The thickness of parts B and C is uniformly distributed with a minimum of 4.98 millimeters and a maximum of 5.02 millimeters. Assume all dimensions are independent.



Develop a spreadsheet model to estimate the probability that the gap  $D$  is less than 10.1 millimeters with 95% confidence to within  $\pm 0.01$  millimeters.

- 3.26 Shipments can be transported by rail or trucks between New York and Los Angeles. Both modes of transport go through the city of St. Louis. The mean travel time

and standard deviations between the major cities for each mode of transportation are shown in the following figure.



Assume that the travel times (in either direction) are lognormally distributed as shown in the figure. For example, the time from New York to St. Louis (or St. Louis to New York) by truck is 30 hours with a standard deviation of 6 hours. In addition, assume that the transfer time in hours in St. Louis is triangularly distributed with parameters (8, 10, 12) for trucks (truck to truck). The transfer time in hours involving rail is triangularly distributed with parameters (13, 15, 17) for rail (rail to rail, rail to truck, truck to rail). We are interested in determining the shortest total shipment time combination from New York to Los Angeles. Develop a spreadsheet simulation for this problem.

- (a) How many shipment combinations are there?
  - (b) Write a spreadsheet expression for the total shipment time of the truck only combination.
  - (c) We are interested in estimating the average shipment time for each shipment combination and the probability that the shipment combination will be able to deliver the shipment within 85 hours.
  - (d) Estimate the probability that a shipping combination will be the shortest.
  - (e) Determine the sample size necessary to estimate the mean shipment time for the truck only combination to within 0.5 hours with 95% confidence.
- 3.27 The times to failure for an automated production process have been found to be randomly distributed according to the Rayleigh distribution:

$$f(x) = \begin{cases} 2\beta^{-2}xe^{-(x/\beta)^2} & x > 0 \\ 0 & \text{otherwise} \end{cases} \quad (3.21)$$

Set up a spreadsheet to generate five random numbers from your algorithm with  $\beta = 2$ .

- 3.28 A firm produces YBox gaming stations for the consumer market. Their profit function is

$$\text{Profit} = (\text{unit price} - \text{unit cost}) \times (\text{quantity sold}) - \text{fixed costs}$$

Suppose that the unit price is \$200 per gaming station and that the other variables have the following probability distributions:

Unit cost	80	90	100	110
Probability	0.20	0.40	0.30	0.10
Quantity sold	1000	2000	3000	
Probability	0.10	0.60	0.30	
Fixed cost	50000	65000	80000	
Probability	0.40	0.30	0.30	

Use a spreadsheet to generate 1000 observations of the profit

- (a) Make a histogram of your observations using your favorite statistical analysis package.
  - (b) Estimate the mean profit from your sample and compute a 99% confidence interval for the mean profit.
  - (c) Estimate the probability that the profit will be positive.
- 3.29 T. Wilson operates a sports magazine stand before each game. He can buy each magazine for 33 cents and can sell each magazine for 50 cents. Magazines not sold at the end of the game are sold for scrap for 5 cents each. Magazines can only be purchased in bundles of 10. Thus, he can buy 10, 20, and so on magazines prior to the game to stock his stand. The lost revenue for not meeting demand is 17 cents for each magazine demanded that could not be provided. Mr. Wilson's profit is as follows:

$$\begin{aligned} \text{Profit} = & (\text{Revenue from sales}) - (\text{Cost of magazines}) \\ & - (\text{Lost profit from excess demand}) \\ & + (\text{Salvage value from sale of scrap magazines}) \end{aligned}$$

Not all game days are the same in terms of potential demand. The type of day depends on a number of factors including the current standings, the opponent, and whether or not there are other special events planned for the game day weekend. There are three types of game days demand: high, medium, and low. The type of day has a probability distribution associated with it.

Type of Day	High	Medium	Low
Probability	0.35	0.45	0.20

The amount of demand for magazines then depends on the type of day according to the following distributions:

Demand	High		Medium		Low	
	PMF	CDF	PMF	CDF	PMF	CDF
40	0.03	0.03	0.1	0.1	0.44	0.44
50	0.05	0.08	0.18	0.28	0.22	0.66
60	0.15	0.23	0.4	0.68	0.16	0.82
70	0.2	0.43	0.2	0.88	0.12	0.94
80	0.35	0.78	0.08	0.96	0.06	1.0
90	0.15	0.93	0.04	1.0		
100	0.07	1.0				

Let  $Q$  be the number of units of magazines purchased (quantity on hand) to set up the stand. Let  $D$  represent the demand for the game day. If  $D > Q$ , Mr. Wilson sells only  $Q$  and will have lost sales of  $D - Q$ . If  $D < Q$ , Mr. Wilson sells only  $D$  and will have scrap of  $Q - D$ . Assume that he has determined that  $Q = 50$ . Make sure that you can estimate the average profit and the probability that the profit is greater than zero for Mr. Wilson. Develop a spreadsheet model to estimate the average profit with 95% confidence to within plus or minus  $\pm \$0.5$ .

- 3.30 The time for an automated storage and retrieval system in a warehouse to locate a part consists of three movements. Let  $X$  be the time to travel to the correct aisle. Let  $Y$  be the time to travel to the correct location along the aisle. And let  $Z$  be the time to travel up to the correct location on the shelves. Assume that the distributions of  $X$ ,  $Y$ , and  $Z$  are as follows:
- $X \sim \text{lognormal}$  with mean 20 and standard deviation 10 seconds
  - $Y \sim \text{uniform}$  with minimum 10 and maximum 15 seconds
  - $Z \sim \text{uniform}$  with minimum of 5 and a maximum of 10 seconds.

Develop a spreadsheet that can estimate the average total time that it takes to locate a part and can estimate the probability that the time to locate a part exceeds 60 seconds. Base your analysis on 1000 observations.

- 3.31 Lead-time demand may occur in an inventory system when the lead-time is other than instantaneous. The lead-time is the time from the placement of an order until the order is received. The lead-time is a random variable. During the lead-time, demand also occurs at random. Lead-time demand is thus a random variable defined as the sum of the demands during the lead-time, or  $\text{LDT} = \sum_{i=1}^T D_i$  where  $i$  is the time period of the lead-time and  $T$  is the lead-time. The distribution of lead-time demand is determined by simulating many cycles of lead-time and the demands that occur during the lead-time to get many realizations of the random variable LDT. Notice that LDT is the *convolution* of a random number of random demands. Suppose that the daily demand for an item is given by the following PMF:

Daily demand (items)	4	5	6	7	8
Probability	0.10	0.30	0.35	0.10	0.15

The lead-time is the number of days from placing an order until the firm receives the order from the supplier.

- (a) Assume that the lead-time is a constant 10 days. Use a spreadsheet to simulate 1000 instances of LDT. Report the summary statistics for the 1000 observations. Estimate the chance that LDT is greater than or equal to 10. Report a 95% confidence interval on your estimate. Use your favorite statistical program to develop a frequency diagram for LDT.
- (b) Assume that the lead-time has a shifted geometric distribution with probability parameter equal to 0.2. Use a spreadsheet to simulate 1000 instances of LDT. Report the summary statistics for the 1000 observations. Estimate the chance that LDT is greater than or equal to 10. Report a 95% confidence interval on your estimate. Use your favorite statistical program to develop a frequency diagram for LDT.
- 3.32 Set up a spreadsheet to generate five random numbers from the negative binomial distribution with parameters ( $r = 4, p = 0.4$ ) using
- the convolution method
  - the number of Bernoulli trials to get four successes.
- 3.33 If  $Z \sim N(0, 1)$  and  $Y = \sum_{i=1}^k Z_i^2$ , then  $Y \sim \chi_k^2$ , where  $\chi_k^2$  is a chi-squared random variable with  $k$  degrees of freedom. Set up a spreadsheet to generate five  $\chi_5^2$  random variates.
- 3.34 Set up a spreadsheet that will generate 30 observations from the following probability density function using the acceptance–rejection algorithm for generating random variates.
- $$f(x) = \begin{cases} \frac{3x^2}{2} & -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

- 3.35 This problem is based on Cheng [1977], see also Ahrens and Dieter [1972]. Consider the gamma distribution:

$$f(x) = \beta^{-\alpha} x^{\alpha-1} \frac{e^{-x/\beta}}{\Gamma(\alpha)}$$

where  $x > 0$  and  $\alpha > 0$  is the shape parameter and  $\beta > 0$  is the scale parameter. In the case where  $\alpha$  is a positive integer, the distribution reduces to the Erlang distribution and  $\alpha = 1$  produces the negative exponential distribution. Acceptance–rejection techniques can be applied to the cases of  $0 < \alpha < 1$  and  $\alpha > 1$ . For the case of  $0 < \alpha < 1$ , see Ahrens and Dieter [1972]. For the case of  $\alpha > 1$ , Cheng [1977] proposed the following majorizing function:

$$g(x) = \left[ \frac{4\alpha^\alpha e^{-\alpha}}{a\Gamma(\alpha)} \right] h(x)$$

where  $a = \sqrt{(2\alpha - 1)}$ ,  $b = \alpha^a$ , and  $h(x)$  is the resulting probability distribution function when converting  $g(x)$  to a density function:

$$h(x) = ab \frac{x^{a-1}}{(b + x^a)^2} \quad \text{for } x > 0$$

- (a) Set up a spreadsheet to generate random variates from a gamma distribution with parameters  $\alpha = 2$  and  $\beta = 10$  via the acceptance/rejection method.

- (b) Use your favorite statistical software to make a histogram of 1000 observations from your implementation.
- 3.36 This procedure is due to Box and Muller [1958]. Let  $U_1$  and  $U_2$  be two independent uniform (0,1) random variables and define

$$X_1 = \cos(2\pi U_2) \sqrt{-2 \ln(U_1)}$$

$$X_2 = \sin(2\pi U_2) \sqrt{-2 \ln(U_1)}$$

It can be shown that  $X_1$  and  $X_2$  will be independent standard normal random variables, that is,  $N(0, 1)$ . Use a spreadsheet to implement the Box and Muller algorithm for generating normal random variables. Generate 1000  $N(\mu = 2, \sigma = 0.75)$  random variables via this method.

- (a) Make a histogram of your observations.  
 (b) Make a normal probability plot of your observations.
- 3.37 The lack of memory property of the exponential distribution states that given  $\Delta t$  is the time period that elapsed since the occurrence of the last event, the time  $t$  remaining until the occurrence of the next event is independent of  $\Delta t$ . This implies that  $P\{X > \Delta t + t | X > t\} = P\{X > t\}$ . Prove that the exponential distribution has the lack of memory property.
- 3.38 Describe a simulation experiment that would allow you to test the lack of memory property of the exponential distribution empirically. See Exercise 3.37. Implement your simulation experiment in a spreadsheet and test the lack of memory property empirically. Explain in your own words what lack of memory means.
- 3.39 Parts arrive to a machine center with three drill presses according to a Poisson distribution with mean  $\lambda$ . The arriving customers are assigned to one of the three drill presses randomly according to the respective probabilities  $p_1, p_2$ , and  $p_3$  where  $p_1 + p_2 + p_3 = 1$  and  $p_i > 0$  for  $i = 1, 2, 3$ . What is the distribution of the interarrival times to each drill press? Specify the parameters of the distribution.
- (a) Suppose that  $p_1, p_2$ , and  $p_3$  equal to 0.25, 0.45, and 0.3, respectively, and that  $\lambda$  is equal to 12 per minute. Set up a spreadsheet to generate the first three arrival times.



---

# 4

---

## INTRODUCTION TO SIMULATION IN ARENA®

### LEARNING OBJECTIVES

- To understand the basic components of the Arena™ environment.
- To be able to perform simple Monte-Carlo simulations in Arena™.
- To be able to recognize and define the characteristics of a discrete-event dynamic system (DEDS).
- To be able to explain how time evolves in a DEDS.
- To be able to develop and read an activity flow diagram.
- To be able to create, run, animate, and examine the results of an Arena™ model of a simple DEDS.

### 4.1 INTRODUCTION

In this chapter, we explore the Arena™ simulation software platform for developing and executing simulation models. After highlighting the modeling environment, we will reconsider some of the examples from Chapter 3 and illustrate how to perform the same simulations using Arena™. This will allow us to introduce the modeling environment without having to introduce new simulation modeling ideas. Then, we will begin our study of the major emphasis of this textbook: modeling discrete-event dynamic systems (DEDSs).

As defined in Chapter 1, a DEDS is a system that evolves dynamically through time. This chapter will introduce how time evolves for DEDSs and illustrate how Arena™ can be used to develop a model for a simple queuing system. Let us jump into Arena™.

## 4.2 THE ARENA ENVIRONMENT

Arena™ is a commercial software program that facilitates the development and execution of computer simulation models. The Arena™ provides access to an underlying simulation language called SIMAN through an environment that permits the building of models using a drag and drop flow chart methodology. The Arena™ environment has panels that provide access to language modeling constructs. In addition, the environment provides toolbar and menu access to common simulation activities, such as animating the model, running the model, and viewing the results.

Figure 4.1 illustrates the Arena™ environment with the View, Draw, Animate, and Animate Transfer toolbars detached (floating) within the environment. In addition, the Project Bar contains the project templates (Advanced Transfer, Advanced Process, Basic Process, Flow Process), the report, and the navigate panels. By right clicking within the project template area, you can access the context menu for attaching/detaching project templates. The project templates, indicated by the diamond flowchart symbol are used to build models in the model window. Normally, the Advanced Transfer and Flow Process panels are not

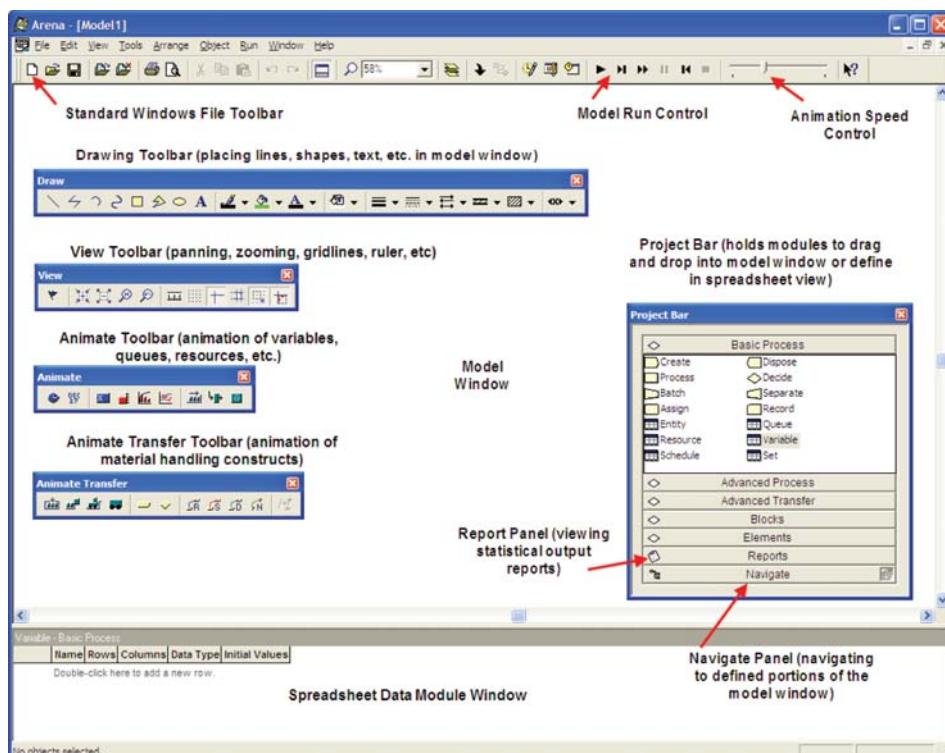


Figure 4.1 Arena™ environment.

attached. Additional modeling constructs can be accessed by attaching these templates to the project bar. In addition, you can control the size and view option for the modules. For example, you can change the view of the modules to see them as large icons, small icons, or text only.

The report panel allows you to drill down into the reports that are generated after a simulation run. The navigation panel allows the definition and use of links (bookmarks) to prespecified areas of the model window. The Arena™ environment normally has the toolbars docked, allowing unobstructed access to the model window and the spreadsheet (data) window. The flow-chart-oriented symbols from the project templates are “dragged and dropped” into the model window. The flow chart symbols are called *modules*.

The spreadsheet view presents a row/column format for the currently selected module in the model window. In addition, the spreadsheet view allows data entry for those modules (e.g., VARIABLE) that are used in the model but for which there is not a flow-chart-oriented representation. You will learn about the various characteristics of flow chart modules and data modules as you proceed throughout this text.

To hide or view the toolbars, you can right-click within the gray toolbar area and check the desired toolbars within the contextual pop-up menu. In addition, you can also drag the toolbars to the desired location within the environment. Figure 4.2 illustrates the Arena™ environment with the toolbars and project bar docked in their typical standard locations. The model window has modules (CREATE, PROCESS, DISPOSE) from the basic process panel and the spreadsheet (data) window is showing the spreadsheet view for the CREATE module. Throughout the text, I will use all capital letters to represent the names of modules. This is only a convention of this textbook to help you to identify Arena™ constructs. We will also use these module names when we write text-based pseudo-code to represent our conceptual understanding of the models.

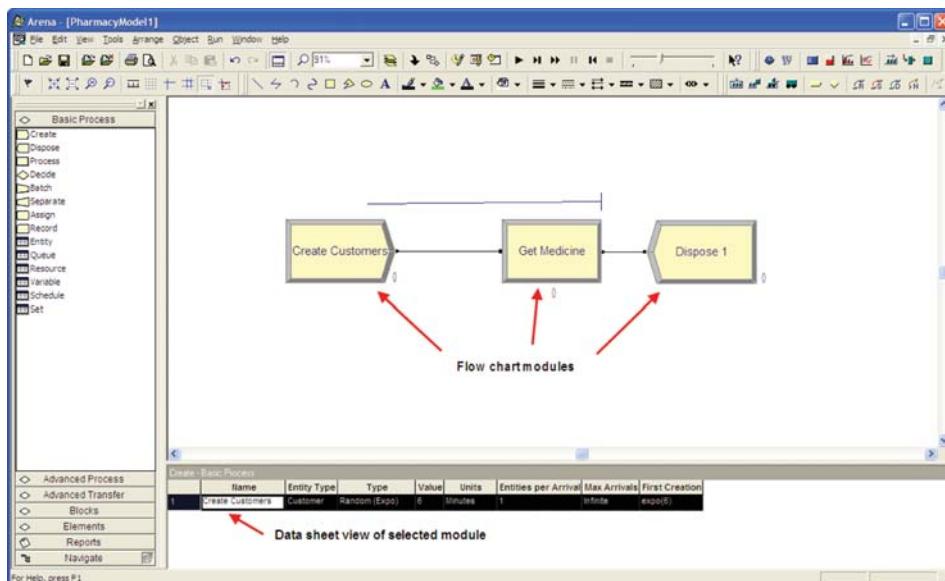


Figure 4.2 Arena™ environment with toolbars docked.

### 4.3 PERFORMING SIMPLE MONTE-CARLO SIMULATIONS USING ARENA

Example models using Arena™ are used throughout this text. The models are supplied with the files that accompany the text. You should explore the completed models as they are discussed within the text; however, in order to get the most out of these examples, you should try to follow along using Arena™ to build the models. In some cases, you can start from scratch. In other cases, a starting model might be given so that you can perform the enhancements. Working through these examples is very important to developing a good understanding of the material. Let us get started working with Arena™.

In this section, we will revisit the Monte-Carlo simulation topic of Chapter 3 and illustrate how the same spreadsheet models can be performed in Arena™. First, we need to introduce four Arena™ programming modules that we will use to develop these simulations.

- **CREATE.** This module creates entities that go through other modules causing their logic to be executed.
- **ASSIGN.** This module allows us to assign values to variables with a model.
- **RECORD.** This module permits the collection of statistics on expressions.
- **DISPOSE.** This module disposes of entities after they are done executing within the model.
- **VARIABLE.** This module is used to define variables within the model and initialize their values.

These modules (VARIABLE, CREATE, ASSIGN, RECORD, and DISPOSE) all have additional functionality that will be described later in the text; however, the above definitions will suffice to get us started for this example.

Second, we need to have a basic understanding of the mathematical and distribution function modeling that is available within Arena™. While Arena™ has a wide range of mathematical and distribution modeling capabilities, only the subset necessary for the Monte-Carlo examples of Chapter 3 will be illustrated in this section.

Here is an overview of the mathematical and distribution functions that we will be using:

- **UNIF(a,b).** This function will return a random variate from a uniform distribution over the range from  $a$  to  $b$ .
- **NORM(mean, std. dev.).** This function returns a random variate from a normal distribution with the specified mean and standard deviation.
- **BETA(alpha1, alpha2).** This function returns a random variate from a beta distribution with shape parameters alpha1 and alpha2.
- **DISC( $cp_1, v_1, cp_2, v_2, \dots$ ).** This function returns a random variate from a discrete empirical cumulative distribution function, where  $cp_i, v_i$  are the pairs  $cp_i = P\{X \leq v_i\}$  for the CDF.
- **SQRT(x).** This function returns the square root of  $x$ .
- **MN(x1, x2,...).** This function returns the minimum of the values presented.
- **MX(x1, x2,...).** This function returns the maximum of the values presented.
- **\*\*.** This operator is used to raise something to a power. For example,  $x ** 3$  is  $x^3$ .

### 4.3.1 Redoing Area Estimation with Arena

Recall Example 3.1. In the example, we are interested in estimating the area under  $f(x) = x^{\frac{1}{2}}$  over the range from 1 to 4. Exhibit 4.1 outlines the basic pseudo-code for estimating the area of the function.

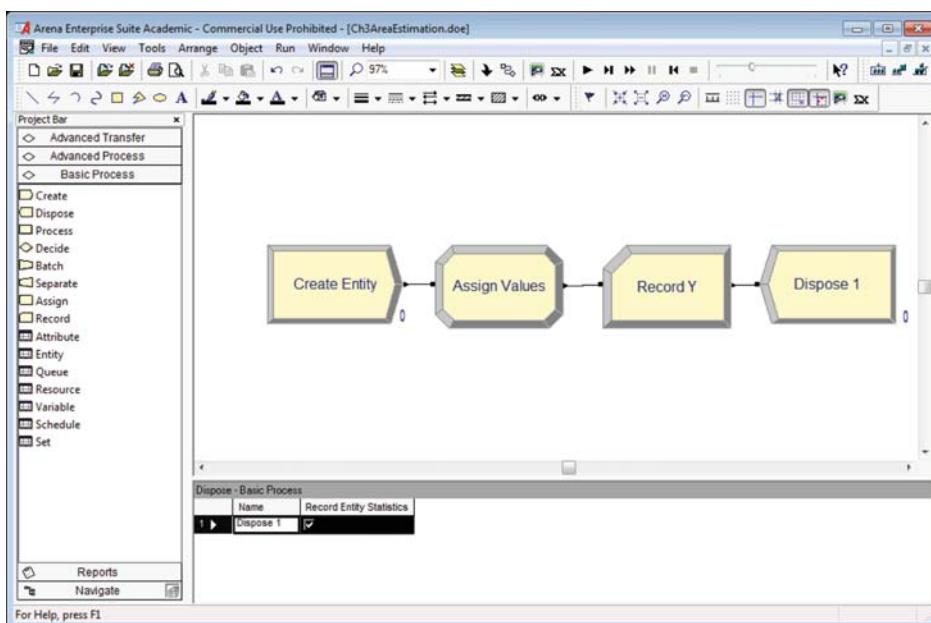
#### Exhibit 4.1 Pseudo-Code for Area Estimation

```

CREATE 1 entity
ASSIGN
    vX1 = UNIF(1,4)
    vFx = SQRT(vX1)
    vY = (4.0 - 1.0)vFx
END ASSIGN
RECORD vY
DISPOSE

```

The CREATE generates 1 entity, which then executes the rest of the code. We will learn more about the concept of an entity later. For the purposes of this example, think of the entity as representing an observation that will invoke the subsequent listed commands. The ASSIGN represents the assignments that generate the value of  $x$ , evaluate  $f(x)$ , and compute  $y$ . The RECORD indicates that statistics should be collected on the value of  $y$ . Finally, the DISPOSE indicates that the generated entity should be disposed (removed from the program). This is only pseudo-code! It does not represent a functioning Arena<sup>TM</sup> model. However, it is useful to conceptualize the program in this manner so that you can more readily implement it within the Arena<sup>TM</sup> environment.



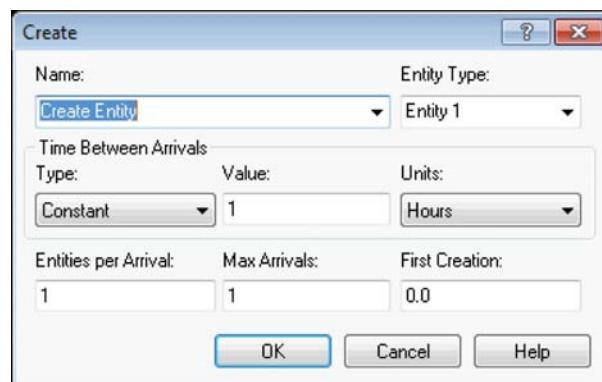
**Figure 4.3** Arena<sup>TM</sup> model for estimating area of curve.

Figure 4.3 represents the flow chart solution within the Arena™ environment. Notice the CREATE, ASSIGN, RECORD, and DISPOSE modules laid out in the model window. A CREATE module creates entities that flow through other modules. Figure 4.4 illustrates the creating of 1 entity by setting the maximum number of entities field to the value 1. Since the First Creation field has a value of 0.0, there will be 1 entity that is created at time 0. The CREATE module will be discussed in further detail later in this chapter. The created entity will move to the next module (ASSIGN) and cause its logic to be executed.

An ASSIGN module permits the assignment of values to various quantities of interest in the model. In Figure 4.5, a listing of the assignments for generating the value of  $x$ , computing  $f(x)$ , and  $y$  are provided. Each assignment is executed in the order listed. It is important to get the order correct when you write your programs!

Figure 4.6 shows the RECORD module for this situation. The value of the variable  $vY$  is in the Value text field. Because the Type of the record is set at Expression, the value of this variable will be observed every time an entity passes through the module. The observations will be tabulated so that the sample average and 95% half-width will be reported automatically on the Arena™ output reports, labeled by the text in the Tally Name field. The results are illustrated in Figure 4.7.

In order to run the model, you must indicate the required number of observations. Since each run of the model produces 1 entity and thus 1 observation, you need to specify the number of times that you want Arena™ to run the model. This is accomplished using the Run Setup menu item. Figure 4.8 shows that the model is set to run 20 replications. Replications will be discussed later in the text, but for now, you can think of a replication as a running



**Figure 4.4** Creating 1 entity with a CREATE module.

Assignments			
	Type	Variable Name	New Value
1	Variable	vXi	UNIF(1,4)
2	Variable	vFx	SQRT(vXi)
3	Variable	vY	(4.0 - 1.0)*vFx
Double-click here to add a new row.			

**Figure 4.5** Making assignments using an ASSIGN module.

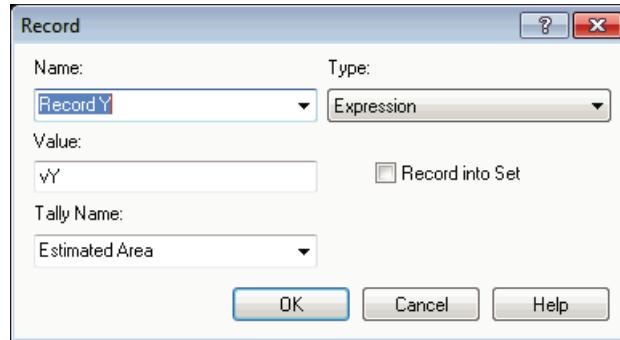


Figure 4.6 Recording statistics on an expression using a RECORD module.

### User Specified

#### Tally

Expression	Average	Half Width
Estimated Area	4.3872	0.39

Figure 4.7 Results from the area estimation model.

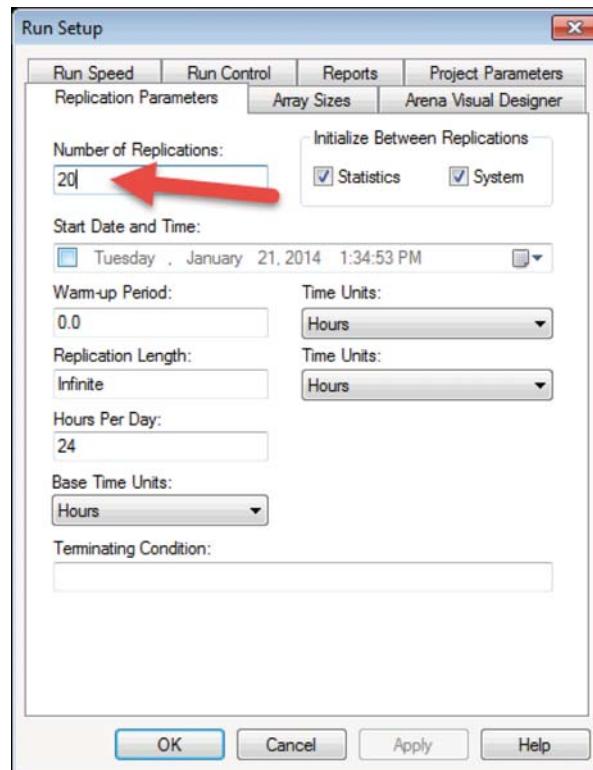


Figure 4.8 Setting up the simulation to replicate 20 observations.

of the model to produce observations. Each replication is independent of other replications. Thus, the observations across replications form a random sample of the observations. Arena™ reports the statistics across the replications.

Figure 4.7 shows the results of estimating the area of the curve. Notice that the average value (4.3872) reported for the variable observed in the RECORD module is close to the true value of 4.66 computed in Chapter 3. As previously discussed, the difference is due to sampling error.

The other Monte-Carlo simulation examples of Chapter 3 have a very similar form.

### 4.3.2 Redoing the News Vendor Problem with Arena

Recall the news vendor problem described in Section 3.3.2. The pseudo-code for the news vendor problem will require the generation of the demand from the following distribution:

$d_i$	5	10	40	45	50	55	60
$f(d_i)$	0.1	0.2	0.3	0.2	0.1	0.05	0.05
$F(d_i)$	0.1	0.3	0.6	0.8	0.9	0.95	1.0

In order to generate from this distribution, the DISC() function should be used. Since the cumulative distribution has already been computed, it is straightforward to write the inputs for the DISC() function. The function specification is

DISC(0.1, 5, 0.3, 10, 0.6, 40, 0.8, 45, 0.9, 50, 0.95, 55, 1.0, 60)

Notice the pairs  $(F(d_i), d_i)$ . The final value of  $F(d_i)$  must be the value 1.0.

First, let us define some variables to be used in the modeling: demand, amount sold, amount left at the end of the day, sales revenue, salvage revenue, price per unit, cost per unit, salvage per unit, and order quantity.

- Let vDemand represent the daily demand.
- Let vAmtSold represent the amount sold.
- Let vAmtLeft represent the amount left at the end of the day
- Let vSalesRevenue represent the sales revenue.
- Let vSalvageRevenue represent the salvage revenue.
- Let vPricePerUnit = 0.25 represent the price per unit.
- Let vCostPerUnit = 0.15 represent the cost per unit.
- Let vSalvagePerUnit = 0.02 represent the salvage value per unit
- Let vOrderQty = 30 represent the amount ordered each day

The pseudo-code for this situation is illustrated in Exhibit 4.2. The CREATE module creates an entity to represent the day. The ASSIGN modules implement the logic associated with the news vendor problem. The MN() function is used to compute the minimum of the demand and the amount ordered. The MX() function determines the amount unsold. The DISC() function implements the CDF of the demand in order to generate the daily demand. The rest of the equations follow directly from the discussion in Chapter 3. The flow chart

**Exhibit 4.2** Pseudo-Code for News Vendor Model

```

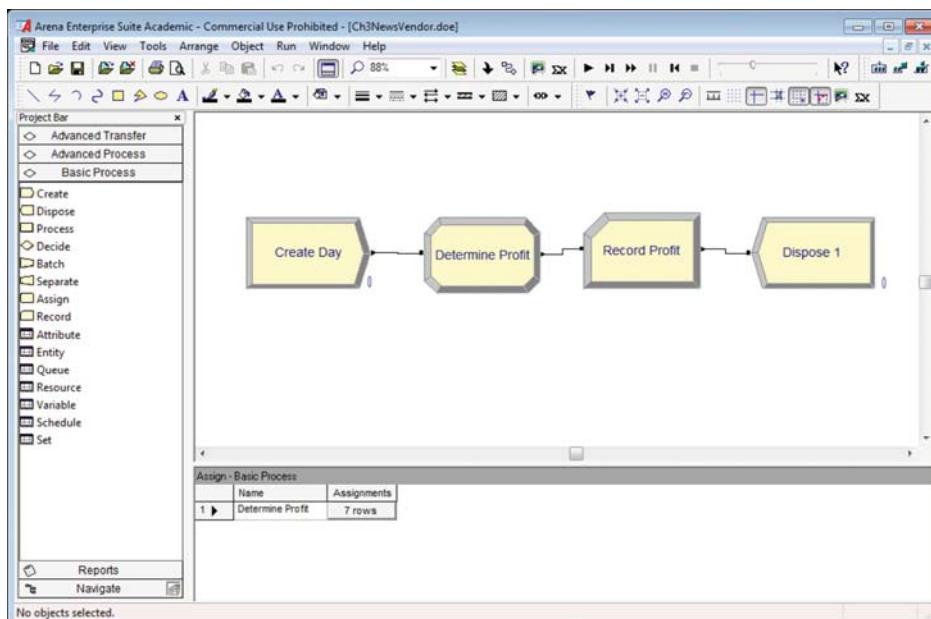
CREATE 1 entity
ASSIGN
    vDemand = DISC(0.1,5,0.3,10,0.6,40,0.8,45,0.9,50,0.95,55,1.0,60)
    vAmtSold = MN(vDemand, vOrderQty)
    vAmtLeft = MX(0, vOrderQty - vDemand)
    vSalesRevenue = vAmtSold*vPricePerUnit
    vSalvageRevenue = vAmtLeft*vSalvagePerUnit
    vOrderingCost = vCostPerUnit*vOrderQty
    vProfit = vSalesRevenue + vSalvageRevenue - vOrderingCost
END ASSIGN
RECORD vProfit
DISPOSE

```

solution is illustrated in Figure 4.9 and uses the same flow chart modules as in the last example.

To implement the variable definitions from the pseudo-code, you need to use a VARIABLE module to provide the initial values of the variables. Notice in Figure 4.10 the definition of each variable and how the initial value of the variable *vCostPerUnit* is showing. The CREATE module is set up exactly as in the last example. The created entity will move to the next module (ASSIGN) illustrated in Figure 4.11. The assignments follow exactly the previously illustrate pseudo-code. Running the model for 100 days results in the output illustrated in Figure 4.12. These results are similar to those computed in Chapter 3.

In this section, we explored how to develop models in Arena<sup>TM</sup> for which time is not a significant factor. In the case of the news vendor problem, where we simulated each day's demand, time advanced at regular intervals. In the case of the area problem, time was not a



**Figure 4.9** Arena<sup>TM</sup> model for news vendor problem.

	Name	Rows	Columns	Data Type	Clear Option	File Name	Initial Values
1 ►	vCostPerUnit			Real	System		1 rows
2	vPricePerUnit			Real	System		1 rows
3	vSalvagePerUnit			Real	System		1 rows
4	vOrderQty			Real	System		1 rows
5	vDemand			Real	System		0 rows
6	vAmtSold			Real	System		0 rows
7	vAmtLeft			Real	System		0 rows
8	vSalesRevenue			Real	System		0 rows
9	vSalvageRevenue			Real	System		0 rows
10	vOrderingCost			Real	System		0 rows
11	vProfit			Real	System		0 rows

Double-click here to add a new row.

**Figure 4.10** Variable definitions used in news vendor solution.

Assignments			
	Type	Variable Name	New Value
1	Variable	vDemand	DISC(0.1, 5, 0.3, 10, 0.6, 40, 0.8, 45, 0.9, 50, 0.95, 55, 1.0, 60)
2	Variable	vAmtSold	MN(vDemand, vOrderQty)
3	Variable	vAmtLeft	MX(0, vOrderQty - vDemand)
4	Variable	vSalesRevenue	vAmtSold*vPricePerUnit
5	Variable	vSalvageRevenue	vAmtLeft*vSalvagePerUnit
6	Variable	vOrderingCost	vCostPerUnit*vOrderQty
7	Variable	vProfit	vSalesRevenue + vSalvageRevenue - vOrderingCost

Double-click here to add a new row

**Figure 4.11** Assignments using the news vendor solution.

### User Specified

#### Tally

Expression	Average	Half Width
Record Profit	1.3440	0.47

**Figure 4.12** Results based on 100 replications of the news vendor model.

factor in the simulation. These types of simulations are often termed static simulation. In the next section, we begin our exploration of simulation where time is an integral component in driving the behavior of the system. In addition, we will see that time will not necessarily advance at regular intervals (e.g., hour 1 and hour 2). This will be the focus of the rest of the textbook.

## 4.4 HOW THE DISCRETE-EVENT CLOCK WORKS

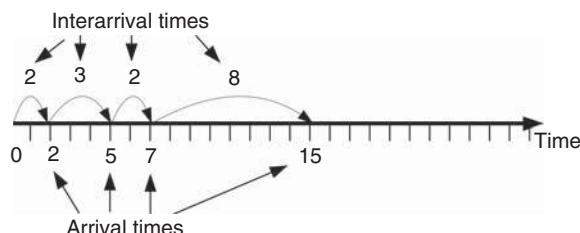
This section introduces the concept of how time evolves in a discrete-event simulation. This topic will also be revisited in future chapters after you have developed a deeper understanding for many of the underlying elements of simulation modeling.

In DEDSs, an event is something that happens at an instant in time which corresponds to a change in system state. An event can be conceptualized as a transmission of information that causes an action resulting in a change in system state. Let us consider a simple bank which has two tellers that serve customers from a single waiting line. In this situation, the system of interest is the bank tellers (whether they are idle or busy) and any customers waiting in the line. Assume that the bank opens at 9 am, which can be used to designate time zero for the simulation. It should be clear that if the bank does not have any customers arrive during the day that the state of the bank will not change. In addition, when a customer arrives, the number of customers in the bank will increase by one. In other words, an arrival of a customer will change the state of the bank. Thus, the arrival of a customer can be considered an event.

Figure 4.13 illustrates a time line of customer arrival events. The values for the arrival times have been rounded to whole numbers, but in general the arrival times can be any real-valued numbers greater than zero. According to the figure, the first customer arrives at time 2 and there is an elapse of 3 min before customer 2 arrives at time five. From the discrete-event perspective, nothing is happening in the bank from time [0,2); however, at time 2, an arrival event occurs and the subsequent actions associated with this event need to be accounted for with respect to the state of the system. Since an event causes actions that result in a change of system state, ask: What are the actions that occur when a customer arrives to the bank?

- The customer enters the waiting line.
- If there is an available teller, the customer will immediately exit the line and the available teller will begin to provide service.
- If there are no tellers available, the customer will remain waiting in the line until a teller becomes available.

Now, consider the arrival of the first customer. Since the bank opens at 9 am with no customer and all the tellers idle, the first customer will enter and immediately exit the queue at time 9:02 am (or time 2) and then begin service. After the customer completes service, the customer will exit the bank. When a customer completes service (and departs the bank),



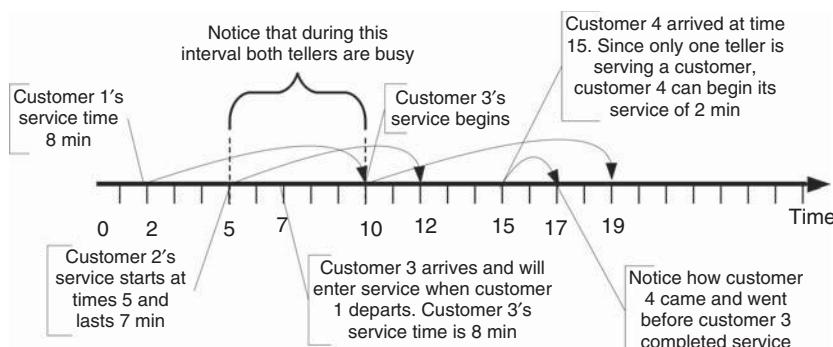
Customer	Time of Arrival	Time between Arrival
1	2	2
2	5	3
3	7	2
4	15	8

Figure 4.13 Customer arrival process.

the number of customers in the bank will decrease by 1. It should be clear that some actions need to occur when a customer completes service. These actions correspond to the second event associated with this system: the customer service completion event. What are the actions that occur at this event?

- The customer departs the bank.
- If there are waiting customers, the teller indicates to the next customer that he/she will serve the customer. The customer will exit the waiting line and will begin service with the teller.
- If there are no waiting customers, then the teller will become idle.

Figure 4.14 contains the service times for each of the four customers who arrived in Figure 4.13. Examine the figure with respect to customer 1. Based on the figure, customer 1 can enter service at time 2 because there were no other customers present in the bank. Suppose that it is now 9:02 am (time 2) and that the service time of customer 1 is known *in advance* to be 8 minutes. From this information, the time at which customer 1 is going to complete service can be precomputed. From the information in the figure, customer 1 will complete service at time 10 (current time + service time =  $2 + 8 = 10$ ). Thus, it should be apparent that for you to recreate the bank's behavior over a time period, you must have knowledge of the customer's service times. The service time of each customer coupled with the knowledge of when the customer began service allows the time that the customer will complete service and depart the bank to be computed in advance. A careful inspection of Figure 4.14 and knowledge of how banks operate should indicate that a customer will begin service either immediately upon arrival (when a teller is available) or coinciding with when another customer departs the bank after being served. This latter situation occurs when the queue is not empty after a customer completes service. The times at which service



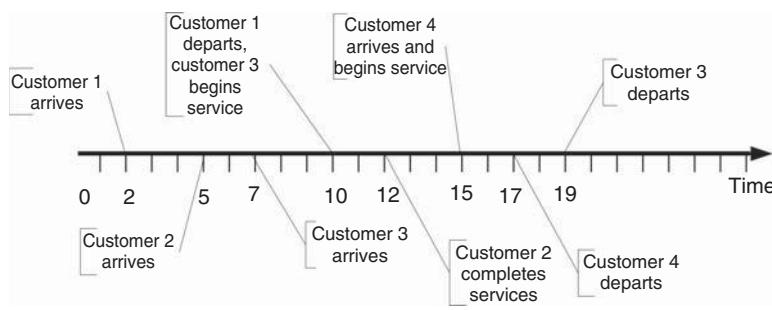
Customer	Service Time Started	Service Time	Service Time Completed
1	2	8	10
2	5	7	12
3	10	9	19
4	15	2	17

Figure 4.14 Customer service process.

completions occur and the times at which arrivals occur constitute the pertinent events for simulating this banking system.

If the arrival and the service completion events are combined, then the time ordered sequence of events for the system can be determined. Figure 4.15 illustrates the events ordered by time from smallest event time to largest event time. Suppose you are standing at time two. Looking forward, the next event to occur will be at time 5 when the second customer arrives. When you simulate a system, you must be able to generate a sequence of events so that at the occurrence of each event, the appropriate actions that change the state of the system are invoked.

The real system will simply evolve over time; however, a simulation of the system must recreate the events. In simulation, events are created by adding additional logic to the normal state changing actions. This additional logic is responsible for scheduling future events that are implied by the actions of the current event. For example, when a customer arrives, the time to the next arrival can be generated and scheduled to occur at some future time. This can be done by generating the time until the next arrival and adding it to the current time to determine the actual arrival time. Thus, all the arrival times do not need to be precomputed prior to the start of the simulation. For example, at time 2, customer 1 arrived. Customer 2 arrives at time 5. Thus, the time between arrivals (3 minutes) is added to the current time



Time	Event	Comment
0	Bank opens	
2	Arrival	Customer 1 arrives, enters service for 8 min, one teller becomes busy
5	Arrival	Customer 2 arrives, enters service for 7 min, the second teller becomes busy
7	Arrival	Customer 3 arrives, waits in queue
10	Service completion	Customer 1 completes service, customer 3 exits the queue and enters service for 9 min
12	Service completion	Customer 2 completes service, no customers are in the queue so a teller becomes idle
15	Arrival	Customer 4 arrives, enters service for 2 min, one teller becomes busy
17	Service completion	Customer 4 completes service, a teller becomes idle
19	Service completion	Customer 3 completes service

Figure 4.15 Order in which events are processed.

and customer 2's arrival at time 5 is scheduled to occur. Every time an arrival occurs, this additional logic is invoked to ensure that more arrivals will continue within the simulation.

Adding additional scheduling logic also occurs for service completion events. For example, since customer 1 immediately enters service at time 2, the service completion of customer 1 can be scheduled to occur at time 12 (current time + service time =  $2 + 10 = 12$ ). Notice that other events may have already been scheduled for times less than time 12. In addition, other events may be inserted before the service completion at time 12 actually occurs. From this, you should begin to get a feeling for how a computer program can implement some of these ideas.

Based on this intuitive notion for how a computer simulation may execute, you should realize that computer logic processing need only occur at the event times. That is, the state of the system does not change between events. Thus, it is *not necessary* to step incrementally through time checking to see if something happens at time 0.1, 0.2, 0.3, etc. (or whatever time scale you envision that is fine enough to not miss any events). Thus, in this book, the simulation clock does not "tick" at regular intervals. Instead, the simulation clock jumps from event time to event time. As the simulation moves from the current event to the next event, the current simulation time is updated to the time of the next event and any changes to the system associated with the next event are executed. This allows the simulation to evolve over time.

## 4.5 MODELING A SIMPLE DISCRETE-EVENT DYNAMIC SYSTEM

This section presents how to model a system in which the state changes at discrete events in time as discussed in the previous section.

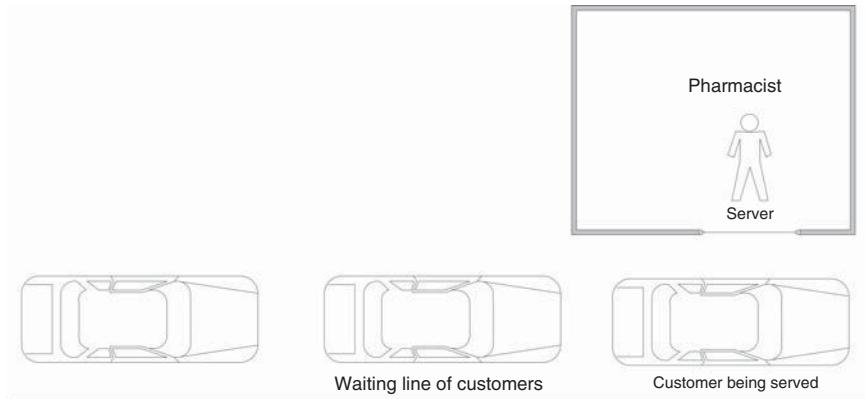
### 4.5.1 A Drive-through Pharmacy

This example considers a small pharmacy that has a single line for waiting customers and only one pharmacist. Assume that customers arrive at a drive-through pharmacy window according to a Poisson distribution with a mean of 10 per hour. The time that it takes the pharmacist to serve the customer is random and data has indicated that the time is well modeled with an exponential distribution with a mean of 3 minutes. Customers who arrive to the pharmacy are served in the order of arrival and enough space is available within the parking area of the adjacent grocery store to accommodate any waiting customers.

The drive-through pharmacy system can be conceptualized as a single server waiting line system, where the server is the pharmacist. An idealized representation of this system is shown in Figure 4.16. If the pharmacist is busy serving a customer, then additional customers will wait in line. In such a situation, management might be interested in how long customers wait in line, before being served by the pharmacist. In addition, management might want to predict if the number of waiting cars will be large. Finally, they might want to estimate the utilization of the pharmacist in order to ensure that he or she is not too busy.

### 4.5.2 Modeling the System

Arena™ is predicated on modeling the process flow of "entities" through a system. Thus, the first question to ask is: *What is the system?* In this situation, the system is the pharmacist and the potential customers as idealized in Figure 4.16. Now you should consider the entities



**Figure 4.16** Drive through pharmacy.

of the system. An entity is a conceptual thing of importance that flows through a system potentially using the resources of the system. Therefore, one of the first questions to ask when developing an Arena™ model is: *What are the entities?* In this situation, the entities are the customers who need to use the pharmacy. This is because customers are discrete things that enter the system, flow through the system, and then depart the system.

Since entities often use things as they flow through the system, a natural question is to ask: *What are the resources that are used by the entities?* A resource is something that is used by the entities and that may constrain the flow of the entities within the system. Another way to think of resources is to think of the things that provide service in the system. In this situation, the entities “use” the pharmacist in order to get their medicine. Thus, the pharmacist can be modeled as a resource.

Since Arena™ facilitates the modeling of the process flows of the entities through a system, it is natural to ask: *What are the process flows?* In answering this question, it is conceptually useful to pretend that you are the entity and to ask: *What do I do?* In this situation, you can pretend that you are a pharmacy customer. *What does a customer do?* Arrive, get served, and leave. As you can see, initial modeling involves identifying the elements of the system and what those elements do. The next step in building an Arena™ model is to enhance your conceptual understanding of the system through conceptual modeling. For this simple system, very useful conceptual model has already been given in Figure 4.16. As you proceed through this text, you will learn other conceptual model building techniques. From the conceptual modeling, you might proceed to more logical modeling in preparation for using Arena™. A useful logical modeling tool is to develop pseudo-code for the situation. Here is some potential pseudo-code for this situation.

- Create customers according to Poisson arrival process
- Process the customers through the pharmacy
- Dispose of the customers as they leave the system.

The pseudo-code represents the logical flow of an entity (customer) through the drive-through pharmacy: Arrive (create), get served (process), and leave (dispose).

Another useful conceptual modeling tool is the *activity diagram*. An *activity* is an operation that takes time to complete. An activity is associated with the state of an object over

an interval of time. Activities are defined by the occurrence of two events which represent the activity's beginning time and ending time and mark the entrance and exit of the state associated with the activity. An activity diagram is a pictorial representation of the process (steps of activities) for an entity and its interaction with resources while within the system. If the entity is a temporary entity (i.e., it flows through the system), the activity diagram is called an activity flow diagram. If the entity is permanent (i.e., it remains in the system throughout its life), the activity diagram is called an activity cycle diagram. The notation of an activity diagram is very simple and can be augmented as needed to explain additional concepts:

**Queues** shown as a circle with queue labeled inside

**Activities** shown as a rectangle with appropriate label inside

**Resources** shown as small circles with resource labeled inside

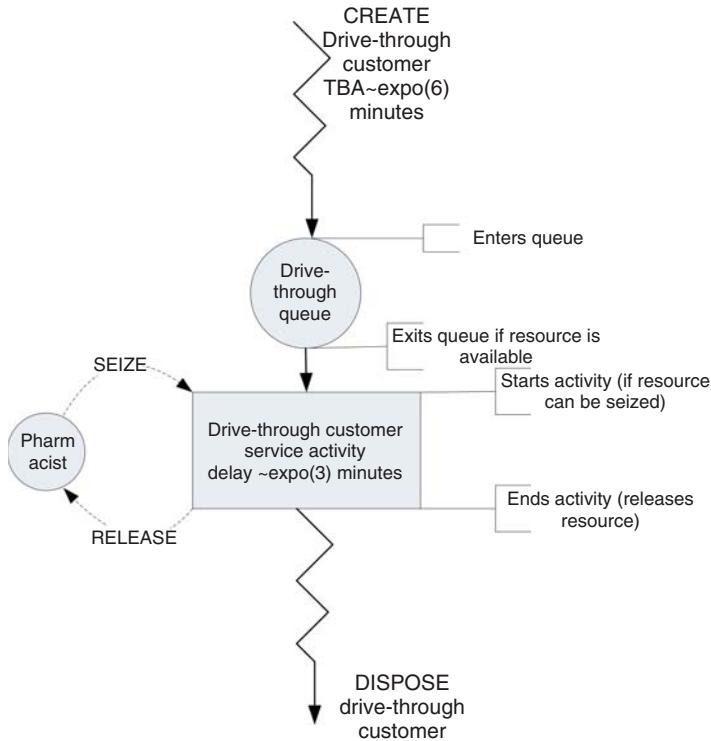
**Lines/arcs** indicating flow (precedence ordering) for engagement of entities in activities or for obtaining resources. Dotted lines are used to indicate the seizing and releasing of resources.

**zigzag lines** indicate the creation or destruction of entities

Activity diagrams are especially useful for illustrating how entities interact with resources. Activity diagrams are easy to build by hand and serve as a useful communication mechanism. Since they have a simple set of symbols, it is easy to use an activity diagram to communicate with people who have little simulation background. Activity diagrams are an excellent mechanism to document your conceptual model of the system before building the model in Arena™.

Figure 4.17 shows the activity diagram for the pharmacy situation. This diagram was built using Visio software, and the drawing is available with the supplemental files for this chapter. You can use this drawing to copy and paste from, in order to form other activity diagrams. The diagram describes the life of an entity within the system. The zigzag lines at the top of the diagram indicate the creation of an entity. While not necessary, the diagram has been augmented with Arena™ like pseudo-code to represent the CREATE statement. Consider following the life of the customer through the pharmacy. Following the direction of the arrows, the customers are first created and then enter the queue. Notice that the diagram clearly shows that there is a queue for the drive-through customers. You should think of the entity flowing through the diagram. As it flows through the queue, the customer attempts to start an activity. In this case, the activity requires a resource. The pharmacist is shown as a resource (circle) next to the rectangle that represents the service activity.

The customer requires the resource in order to start its service activity. This is indicated by the dashed arrow from the pharmacist (resource) to the top of the service activity rectangle. If the customer does not get the resource, they wait in the queue. Once they receive the number of units of the resource requested, they proceed with the activity. The activity represents a delay of time and in this case the resource is used throughout the delay. After the activity is completed, the customer releases the pharmacist (resource). This is indicated by another dashed arrow, with the direction indicating that the resource is “being put back” or released. After the customer completes their service activity, the customer leaves the system. This is indicated with the zigzag lines going to nowhere and augmented with the keyword DISPOSE. The dashed arrows of a typical activity diagram have also been



**Figure 4.17** Activity diagram of drive-through pharmacy.

augmented with the Arena™ like pseudo-code of SEIZE and RELEASE. The conceptual model of this system can be summarized as follows:

**System** The system has a pharmacist who acts as a resource, customers who act as entities, and a queue to hold the waiting customers. The state of the system includes the number of customers in the system, the queue, and service.

**Events** Arrivals of customers to the system, which occur within an interevent time that is exponentially distributed with a mean of 6 minutes.

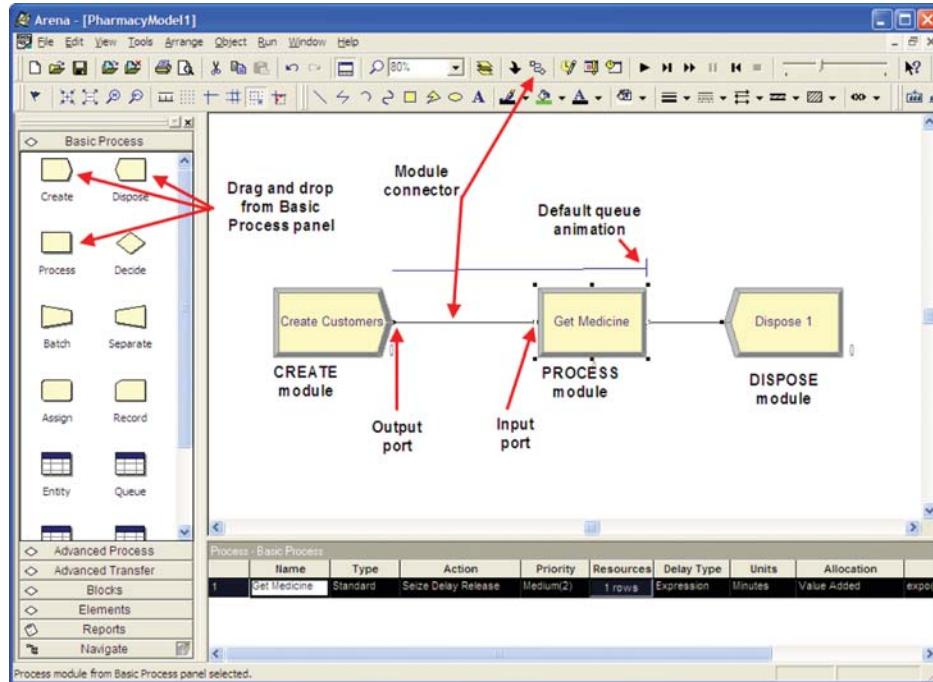
**Activities** The service time of the customers are exponentially distributed with a mean of 3 minutes.

**Conditional delays** A conditional delay occurs when an entity has to wait for a condition to occur in order to proceed. In this system, the customer may have to wait in a queue until the pharmacist becomes available.

With an activity diagram and pseudo-code such as this available to represent a solid conceptual understanding of the system, you can begin the model development process in Arena™.

### 4.5.3 Implementing the Model in Arena

Now, you are ready to implement the conceptual model within Arena™. If you have not already done so, open up the Arena™ environment. Using the Basic Process Panel template,



**Figure 4.18** Overview of pharmacy model.

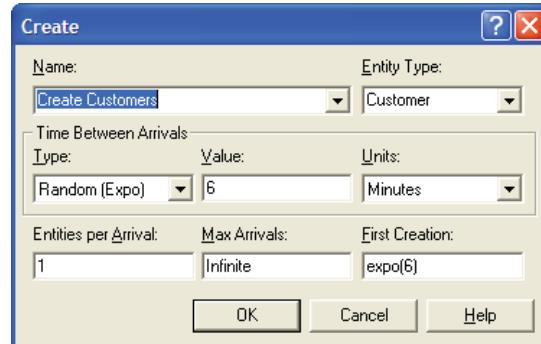
drag the CREATE, PROCESS, and DISPOSE modules into the model window and make sure that they are connected as shown in Figure 4.18. In order to drag and drop the modules, simply select the module within the Basic Process template and drag the module into the model window, letting go of the mouse when you have positioned the module in the desired location. If the module within the model window is highlighted (selected), the next dragged module will automatically be connected. If you placed the modules and they were not automatically connected, then you can use the connect toolbar icon. Select the icon and then select the “from” module near the connection exit point and then holding the mouse button down drag to form a connection line to the entry point of the “to” module. The names of the modules will not match what is shown in the figure. You will name the modules as you fill out their dialog boxes.

#### 4.5.4 Specify the Arrival Process

Within Arena™ nothing happens unless entities enter the model. This is done through the use of the CREATE module. In the current example, pharmacy customers arrive according to a Poisson process with a mean of  $\lambda = 10$  per hour. According to probability theory, this implies that the time between arrivals is exponentially distributed with a mean of  $(1/\lambda)$ . Thus, for this situation, the mean time between arrivals is 6 minutes.

$$\frac{1}{\lambda} = \frac{1 \text{ hour}}{10 \text{ customers}} \times \frac{60 \text{ minutes}}{1 \text{ hour}} = \frac{6 \text{ minutes}}{\text{customers}} \quad (4.1)$$

Open up the CREATE module (by double clicking on it) and fill it in as shown in Figure 4.19. The distribution for the “Time Between Arrivals” is by default Exponential,



**Figure 4.19** CREATE module.

Random (expo) in the figure. In this instance, the “Value” text box refers to the mean time between arrivals. “Entities per Arrival” specifies how many customers arrive at each arrival. Since more than one customer does not arrive at a time, the value is set to 1. “Max Arrivals” specifies the total number of arrival events that will occur for the CREATE module. This is set at the default “Infinite” since a fixed number of customers to arrive is not specified for this example. The “First Creation” specifies the time of the first arrival event. Technically, the time to the first event for a Poisson arrival process is exponentially distributed. Hence, expo(6) has been used with the appropriate mean time, where  $\text{expo}(\text{mean})$  is a function within Arena<sup>TM</sup> that generates random variables according to the exponential distribution function with the supplied mean.

Make sure that you specify the units for time as minutes and be sure to press OK; otherwise, your work within the module will be lost. In addition, as you build models you never know what could go wrong; therefore, you should save your models often as you proceed through the model building process. Take the opportunity to save your model before proceeding.

Before proceeding, there is one last thing that you should do related to the entities. Since the customers drive cars, you will change the picture associated with the customer entity that was defined in the CREATE module. To do this, you will use the ENTITY module within the Basic Process panel. The ENTITY module is a data module. A data module cannot be dragged and dropped into the model window. Data modules require the model builder to enter information in either the spreadsheet window or a dialog box. To see the dialog box, select the row from the spreadsheet view, right-click, and choose the edit via dialog option. You will use the spreadsheet view here. Select the ENTITY module in the Basic Process panel and use the corresponding spreadsheet view to select the picture for the entity as shown in Figure 4.20.

Entity - Basic Process					
	Entity Type	Initial Picture	Holding Cost / Hour	Initial VA Cost	Initial NVA Cost
1	Customer	Picture.Van	0.0	0.0	0.0
Double-click here to add a new row.					

**Figure 4.20** ENTITY module.

#### 4.5.5 Specify the Resources

Go to the Basic Process Panel and select the RESOURCE module. This module is also a data module. As you selected the RESOURCE module, the spreadsheet window should have changed to reflect this selection (Figure 4.21). Double-click on the row in the spreadsheet module for the RESOURCE module to add a resource to the model.

After the resource row has been added, select the row and right-click. This will bring up a context menu. From this context menu, select edit via dialog box. Make the resulting dialog box look like Figure 4.22. You can also type in the same information that you typed in the dialog box from within the spreadsheet view. This defines a resource that can be used in the model. The resource's name is Pharmacist and it has a capacity of 1. Now, you have to indicate how the customers will use this resource within a process.

#### 4.5.6 Specify the Process

A process can be thought of as a set of activities experienced by an entity. There are two basic ways in which processing can occur: resource constrained and resource unconstrained. In the current situation, because there is only one pharmacist and the customer will

Resource - Basic Process					
Name	Type	Busy / Hour	Idle / Hour	Per Use	Report Statistics
Double-click here to add a new row.					

Figure 4.21 RESOURCE data module in spreadsheet window.

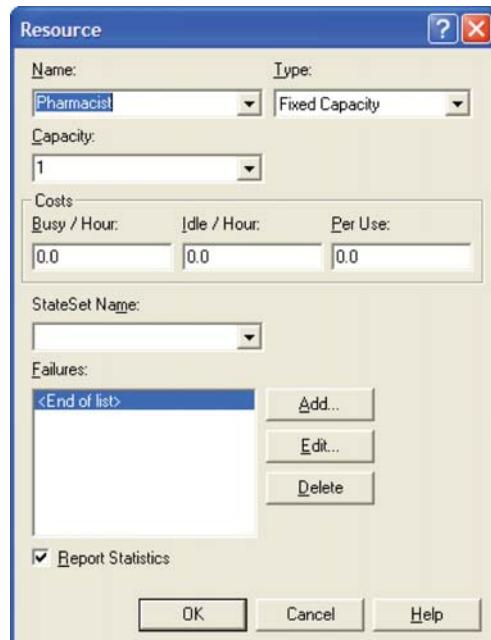


Figure 4.22 RESOURCE dialog box.

need to wait if the pharmacist is busy, the situation is resource constrained. A PROCESS module provides for the basic modeling of processes within an Arena™ model. You specify this within a PROCESS module by having the entity *seize* the resource for the specified usage time. After the usage time has elapsed, you specify that the resource is *released*. The basic pseudo-code can now be modified to capture this concept as illustrated in Exhibit 4.3.

---

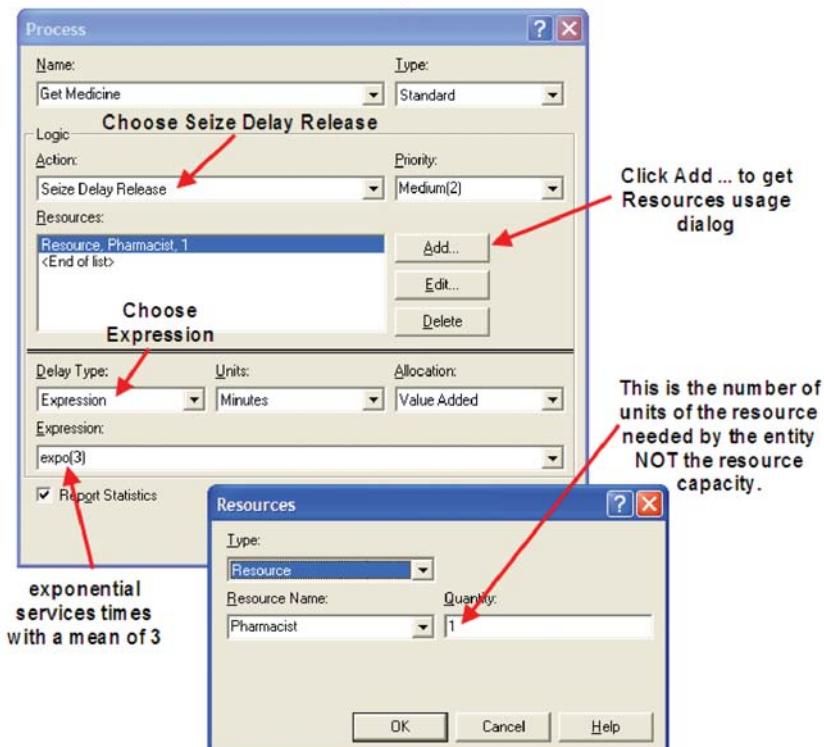
**Exhibit 4.3** Pseudo-code for Pharmacy Model

---

```
CREATE customers with EXPO(6) time between arrivals
PROCESS customers
    SEIZE 1 pharmacist
    DELAY for EXPO(3) minutes
    RELEASE 1 pharmacist
END PROCESS
DISPOSE customer
```

---

Open up the PROCESS module and fill it out as indicated in Figure 4.23. Change the “Action” drop down dialog box to the SEIZE, DELAY, RELEASE option. Use the “Add” button within the “Resources” area to indicate which resources the customer will use. In the pop-up Resources dialog, indicate the name of the resource and the number of units desired



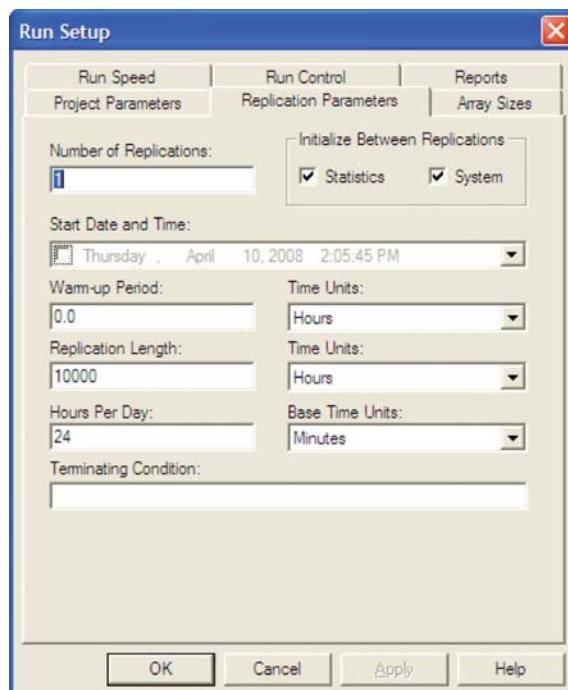
**Figure 4.23** PROCESS module Seize, Delay, Release option.

by the customer. Within the “Delay Type” area, choose Expression as the type of delay and type in  $\text{expo}(3)$  as the expression. This indicates that the delay, which represents the service time, will be randomly distributed according to an exponential distribution with a mean of 3 minutes. Make sure to change the units accordingly. Now you are ready to specify how to run the model.

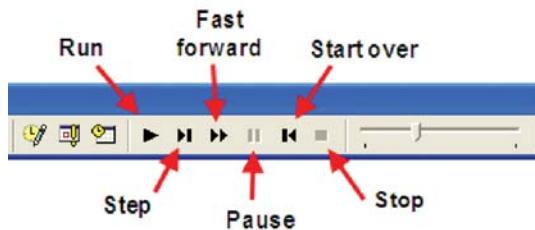
#### 4.5.7 Specify Run Parameters

Let us assume that the pharmacy is open 24 hours a day, 7 days a week. In other words, it is always open. In addition, assume that the arrival process does not vary with respect to time. Finally, assume that management is interested in understanding the long-term behavior of this system in terms of the average waiting time of customers, the average number of customers, and the utilization of the pharmacist.

To simulate this situation over time, you must specify how long to run the model. Ideally, since management is interested in long run performance, you should run the model for an infinite amount of time to get long-term performance; however, you probably do not want to wait that long! For the sake of simplicity, assume that 10,000 hours of operation is long enough. Within the Arena™ environment, go to the Run menu item and choose Setup. After the Setup dialog box appears, select the Replication parameters tab and fill it out as shown in Figure 4.24. The “Replication Length” text box specifies how long the simulation will run. Notice that the base time units were changed to minutes. This ensures that information reported by the model is converted to minutes in the output reports.



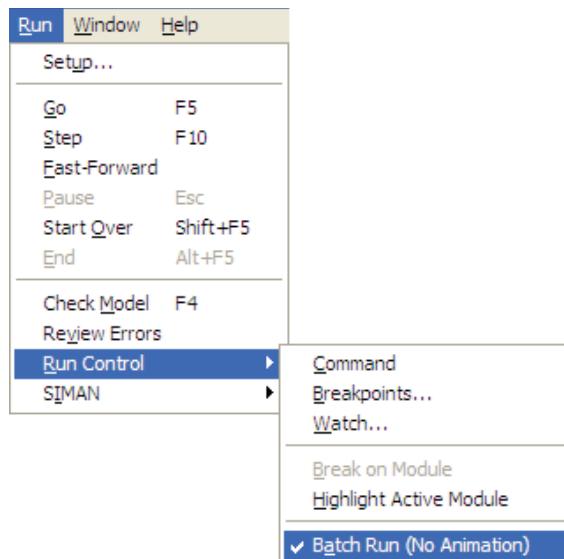
**Figure 4.24** Run Setup Replication Parameters tab.



**Figure 4.25** Run toolbar.

Now, the model is ready to be executed. You can use the Run menu to do this or you can use the convenient “VCR” like run toolbar (Figure 4.25). The run button causes the simulation to run until the stopping condition is met. The fast forward button runs the simulation without animation until the stopping condition is met. The pause button suspends the simulation run. The start over button stops the run and starts the simulation again. The stop button causes the simulation to stop. The animation slider causes the animation to slow down (to the left) or speed up (to the right).

When you run the model, you will see the animation related to the customers waiting in line. Because the length of this run is very long, you should use the fast forward button on the “VCR” run toolbar to execute the model to completion without the animation. Instead of using the fast forward button, you can significantly speed up the execution of your model by running the model in batch mode without animation. To do this, you can use the Run menu as shown in Figure 4.26 and then when you use the VCR Run button, the model will run much faster without animation.

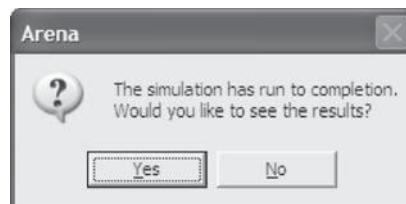


**Figure 4.26** Batch Run No Animation option.

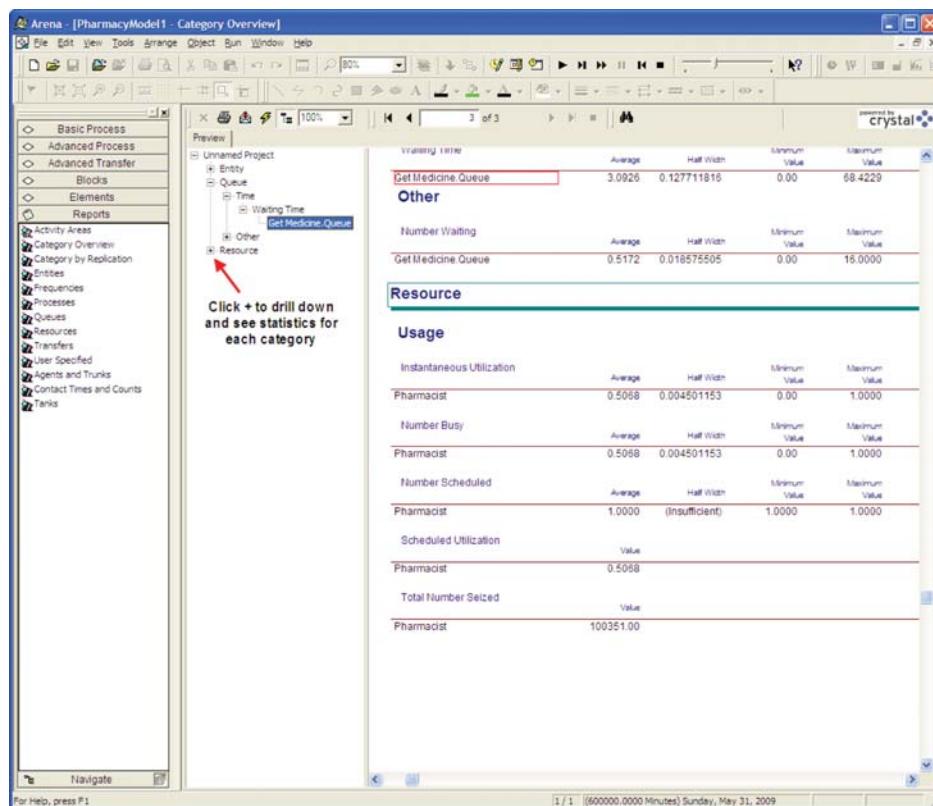
#### 4.5.8 Analyze the Results

After running the model, you will see a dialog indicating that the simulation run is completed and that the simulation results are ready (Figure 4.27). Answer yes to see open up the report viewer. Arena™ writes the simulation output to a database and then uses Crystal Reports to prepare standardized reports.

In the reports preview area, you can drill down to see the statistics that you need. Go ahead and do this so that you have the report window as indicated in Figure 4.28. The reports indicate that customers wait about 3 minutes on an average in the line. Arena™ reports the average of the waiting times in the queue as well as the associated 95% confidence interval half-width. In addition, Arena™ reports the minimum and maximum of the



**Figure 4.27** Run completion dialog.



**Figure 4.28** Results for pharmacy model.

observed values. These results indicate that the maximum a customer waited to get service was 16 minutes. The utilization of the pharmacist is about 50%. This means that about 50% of the time the pharmacist was busy. For this type of system, this is probably not a bad utilization, considering that the pharmacist probably has other in-store duties. The reports also indicate that there was less than one customer on an average waiting for service. Using the Reports panel in the Project Bar, you can also select specific reports.

Arena™ also produces a text-based version of these reports in the directory associated with the model. Within the Windows File Explorer, select the *modelname.out* file, see Figure 4.29. This file can be read with any text editor such as Notepad, see Figure 4.30.

Also as indicated in Figure 4.29, Arena™ generates additional files when you run the model. The *modelname.mdb* file is a Microsoft Access database that holds the information displayed by Crystal Reports. The *modelname.p* file is generated when the model is checked or run. If you have errors or warnings when you check your model, the error file will also show up in the directory of your *modelname.doe* file.

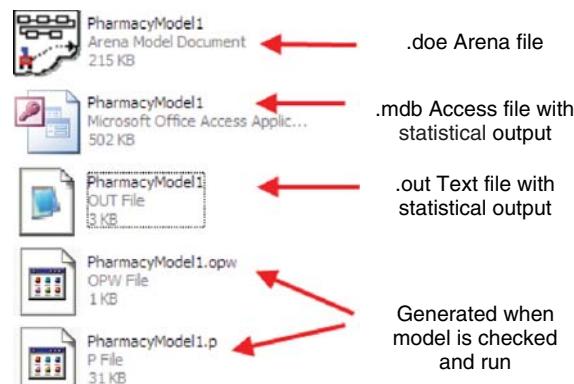
This single server waiting line system is a very common situation in practice. In fact, this exact situation has been studied mathematically through a branch of operations research called queuing theory. As will be covered in Chapter 8, for specific modeling situations, formulas for the long-term performance of queuing systems can be derived. This particular pharmacy model happens to be an example of an M/M/1 queuing model. The first M stands for Markov arrivals, the second M stands for Markov service times, and the 1 represents a single server. Markov was a famous mathematician who examined the exponential distribution and its properties. According to queuing theory, the expected number of customer in queue,  $L_q$ , for the M/M/1 model is

$$L_q = \frac{\rho^2}{1 - \rho} \quad (4.2)$$

$$\rho = \lambda / \mu \quad (4.3)$$

$\lambda$  = Arrival rate to queue (4.4)

$\mu$  = Service rate (4.5)



**Figure 4.29** Files generated when running Arena™.

```

PharmacyModel1 - Notepad
File Edit Format View Help

ARENA Simulation Results
Industrial Engineering

Summary for Replication 1 of 1

Project: Unnamed Project
Analyst: Run execution date : 4/10/2008
Replication ended at time : 600000.0 Minutes Model revision date: 4/10/2008
Base Time Units: Minutes

TALLY VARIABLES
Identifier Average Half width Minimum Maximum Observations
Customer.VATime 3.0301 .02175 2.0142E-05 37.451 100350
Customer.NVATime .00000 .00000 .00000 .00000 100350
Customer.WaitTime 3.0926 .12771 .00000 68.422 100350
Customer.TranTime .00000 .00000 .00000 .00000 100350
Customer.OtherTime .00000 .00000 .00000 .00000 100350
Customer.TotalTime 6.1227 .14416 2.0142E-05 70.525 100350
Get Medicine.Queue.WaitingTime 3.0925 .12771 .00000 68.422 100351

DISCRETE-CHANGE VARIABLES
Identifier Average Half width Minimum Maximum Final value
Customer.WIP 1.0240 .02218 .00000 17.000 1.0000
Pharmacist.NumberBusy .50679 .00450 .00000 1.0000 1.0000
Pharmacist.Numberscheduled 1.0000 (Insuf) 1.0000 1.0000 1.0000
Pharmacist.Utilization .50679 .00450 .00000 1.0000 1.0000
Get Medicine.Queue.NumberInqueue .51724 .01858 .00000 16.000 .00000

OUTPUTS
Identifier Value
Customer.NumberIn 1.0035E+05
Customer.NumberOut 1.0035E+05
Pharmacist.NumberSeized 1.0035E+05
Pharmacist.Scheduledutilization .50679
System.NumberOut 1.0035E+05

Simulation run time: 0.02 minutes.
Simulation run complete.

```

**Figure 4.30** Text file summary results.

In addition, the expected waiting time in queue is given by  $W_q = L_q/\lambda$ . In the pharmacy model,  $\lambda = 1/6$ , that is, one customer for every 6 minutes on an average, and  $\mu = 1/3$ , that is, one customer for every 3 minutes on an average. The quantity,  $\rho$ , is called the utilization of the server. Using these values in the formulas for  $L_q$  and  $W_q$  results in

$$\rho = 0.5$$

$$L_q = \frac{0.5 \times 0.5}{1 - 0.5} = 0.5$$

$$W_q = \frac{0.5}{1/6} = 3 \text{ minutes}$$

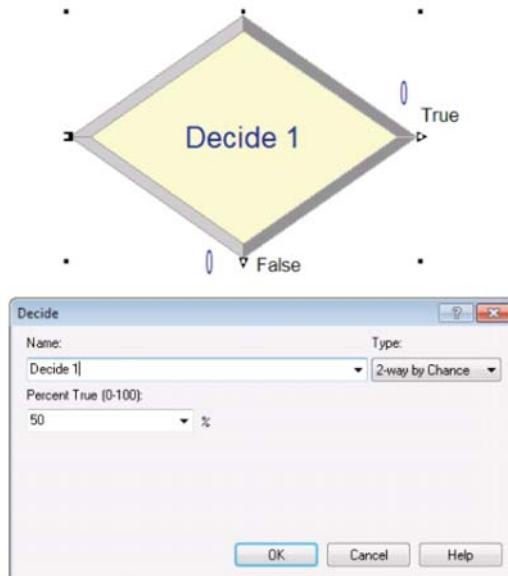
In comparing these analytical results with the simulation results, you can see that they match to within statistical error. Later in this text, the analytical treatment of queues and the simulation of queues will be developed. These analytical results are available for this special case because the arrival and service distributions are exponential; however, simple analytical results are not available for many common distributions, for example, log-normal. With simulation, you can easily estimate the above quantities as well as many other performance measures of interest for wide ranging queuing situations. For example,

through simulation you can easily estimate the chance that there are three or more cars waiting.

#### 4.6 EXTENDING THE DRIVE THROUGH PHARMACY MODEL

The drive-through pharmacy model presents a very simple process for the customer: enter, get served, and depart. To make the model a little more realistic, consider that a customer may decide not to wait in line if there is a long line of waiting customers. Let us assume that if there are four or more customers in the line when a customer arrives, they decide to go to different pharmacy. To model this situation, we need to be able to direct the customer along different paths in the model. This can be accomplished using the DECIDE module. The DECIDE module shown in Figure 4.31 has one input port and possibly two or more output ports. Thus, an entity that enters in the input port side of the DECIDE module can take different paths out of the module. The path can be chosen based on a condition or set of conditions or based on a probability distribution.

To model the situation that an arriving pharmacy customer may decide to go to a different pharmacy, we need to use a condition within the DECIDE module. The decision to go to a different pharmacy depends on how many customers are in the waiting line. Thus, we need a method to determine how many customers are in the pharmacy queue at any time. This can be accomplished using the  $NQ(queue\ name)$  function. The  $NQ(queue\ name)$  function returns the number of entities being held in the named queue. In the model, the queue that holds the customers is call *Get Medicine.Queue*. The name of the queue for a PROCESS module takes on the name of the PROCESS module with the word Queue appended. Therefore, the following condition can be used to test whether or not the arriving customer should go somewhere else is  $NQ(Get\ Medicine.Queue) <= 3$ . The adjusted pseudo-code is shown in Exhibit 4.4.



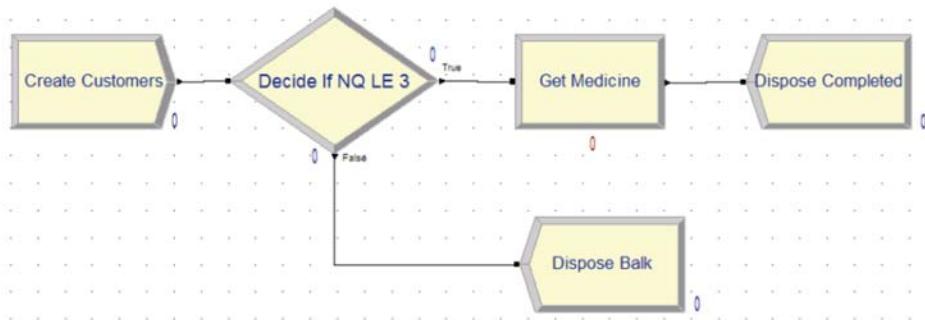
**Figure 4.31** DECIDE flowchart symbol and module dialog.

**Exhibit 4.4** Pseudo-code with DECIDE for Pharmacy Model

```

CREATE customers with EXPO(6) time between arrivals
DECIDE If NQ(Get Medicine.Queue) <= 3
    PROCESS customers
        SEIZE 1 pharmacist
        DELAY for EXPO(3) minutes
        RELEASE 1 pharmacist
    END PROCESS
END DECIDE
DISPOSE customer

```



**Figure 4.32** Overview of pharmacy model with DECIDE module.



**Figure 4.33** DECIDE module dialog with condition specified.

Based on this pseudo-code, the customer only enters the PROCESS module if there are three or less cars waiting in line. The overall model is illustrated in Figure 4.32. Notice that there are now two DISPOSE modules, one for customers who do not enter (balk) and one for those who enter and complete service. Figure 4.33 shows the use of the NQ() function to chose the correct path.

Running the model using the same conditions as before results less waiting and utilization as show in Figure 4.34. This is because less customers enter the system. In the next chapter, we will learn how to quantify the probability of losing customers due to long lines.

<b>Time</b>		
Waiting Time	Average	Half Width
Get Medicine.Queue	2.5724	0.077521344
<b>Other</b>		
Number Waiting	Average	Half Width
Get Medicine.Queue	0.4213	0.011593416
<b>Resource</b>		
<b>Usage</b>		
Instantaneous Utilization	Average	Half Width
Pharmacist	0.4943	0.004910555
Number Busy	Average	Half Width
Pharmacist	0.4943	0.004910555

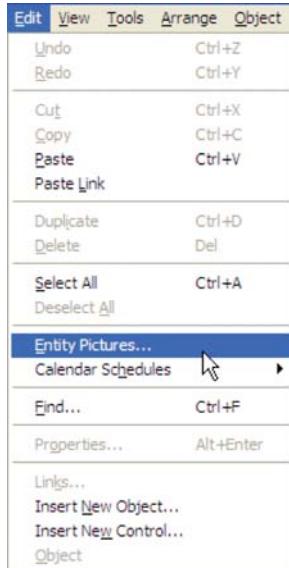
**Figure 4.34** Results for pharmacy model with DECIDE module.

## 4.7 ANIMATING THE DRIVE-THROUGH PHARMACY MODEL

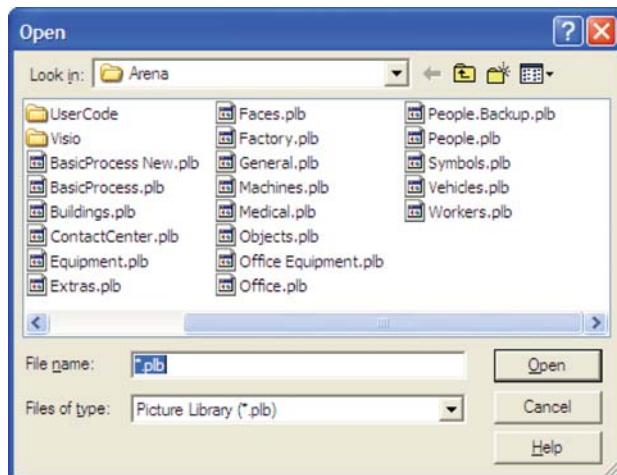
While running the pharmacy model with the animation on, you saw some of Arena's basic animation capabilities. Positioned over the PROCESS module is an animation queue, which shows the cars that are waiting for the pharmacist. Arena™ has many other animation features that can be added to a model. Animation is important for helping to verify that the model is working as intended and for validating the model's representation of the real system. This section illustrates a few of Arena's simpler animation features by having you augment the basic pharmacy model. In particular, you will change the entity picture from a van to a car, draw the pharmacy building, and show the state of the pharmacist (idle or busy). You will also animate the number of cars waiting in the line (both visually and numerically).

When defining the entity type, a picture of a van was originally selected for the entity. Arena™ has a number of other predefined graphical pictures that can be used within the model. The available entity pictures can be found from the Edit menu as shown in Figure 4.35.

Once you have selected Edit > Entity Pictures, you should see the entity picture placement dialog as shown in Figure 4.37. The entity picture placement dialog is divided into two functions: the picture list (on the left of dialog) and the picture library (on the right of the dialog). The picture list represents the entity pictures that are listed in the ENTITY module. If you scroll down, you will find the picture of the van in the list. By navigating to the Arena™ picture libraries within the Arena™ distribution folder on your hard-drive



**Figure 4.35** Accessing entity pictures.



**Figure 4.36** Arena™ picture library files.

(see Figure 4.36), you can select the *Vehicles.plb* file. When selected as the current library, your entity picture placement dialog should look as shown in Figure 4.37.

Scroll down in the vehicle library and select the red car, then select the “<<” button to move the picture to the picture list. Finally, you should give the picture a name in the list (e.g., Red Car). Now, go back to the ENTITY module and associate the picture with your entity. At the time of this writing, the name does not appear on the Entity module drop down list, but the picture is available. Just type in the correct name in the Entity module to have the entity picture represented by the red car.

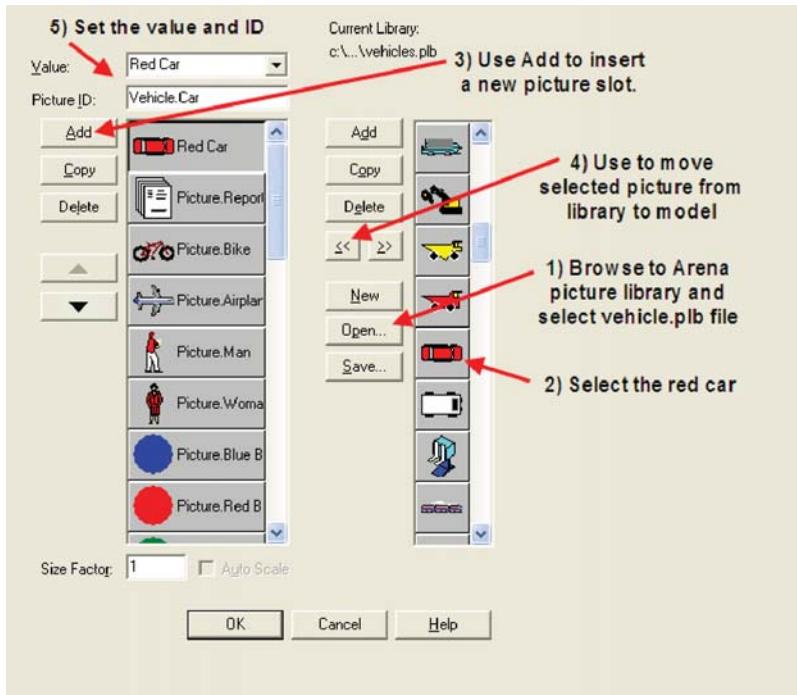


Figure 4.37 Entity picture placement dialog.

Now, you will use Arena's drawing tools to draw the outline of the pharmacy building. Arena™ has standard drawing tools (e.g. lines, rectangles, and polygons) as well as way to add text, fill options etc. These tools are available through the Drawing toolbar as shown in Figure 4.38. You can also turn on the drawing grid and the ruler, which are useful in placing the drawing objects. In this situation, drawing will be kept to a rather simple/crude representation of the pharmacy. Essentially, the pharmacy will be represented as a box, the drive-through lane as a line with the road line pattern, and the pass through service window as two lines with 3pt thickness.

Figure 4.39 illustrates the background drawing for the pharmacy. The Arena™ drawing editor works like most computer-based drawing editors. You should just drag and drop the items and set the fill options, thickness, and line pattern as you go. This portion of this example is available in the *PharmacyModel1WithAnimationSI.doe* file.

In the next part of the example, you will animate the resource so that you can see when the pharmacist is busy or idle, provide a location to show the customer currently being served, and better represent the waiting cars in the drawing.

Within Arena™, a resource can be in one of four default states (idle, busy, inactive, and failed). The inactive and failed states will be discussed later in the text. The idle state represents the situation that at least 1 unit of the resource is available. In the pharmacy model, there is only one pharmacist, so if the pharmacist is not serving the customer, the pharmacist is considered idle. The busy state represents the situation where all available units of the resource are currently seized. In the case of the pharmacy model, since there is only one resource unit (the pharmacist), the resource is busy when the sole pharmacist is

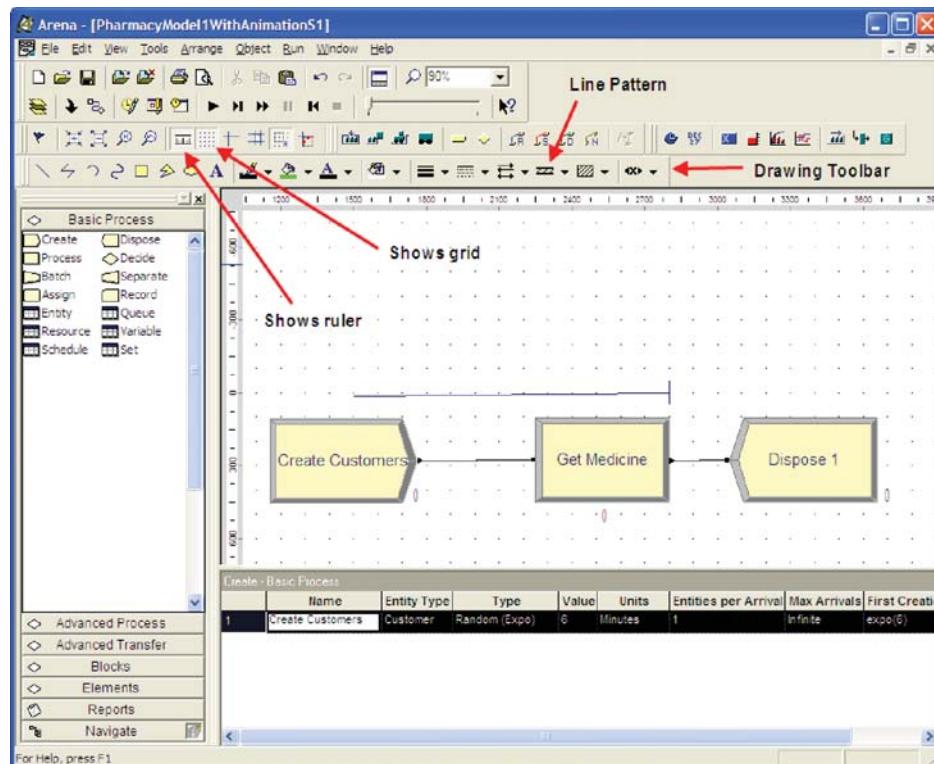


Figure 4.38 Basic drawing tools in Arena™.

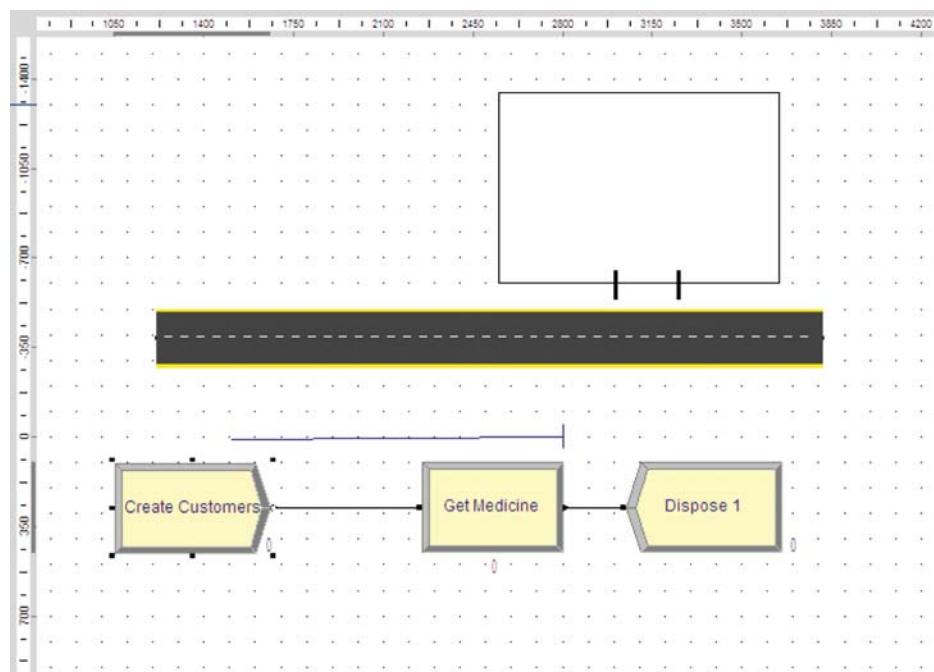


Figure 4.39 Simple pharmacy drawing in Arena™.

seized. In the animation, you will visibly represent the state of the pharmacist. This can be done through Arena's animate toolbar and in particular the animate resource button.

1. Click the Resource button on the Animate toolbar.
2. The Resource Placement dialog box appears. Use the open button to navigate to the *people.plb* picture library (e.g., C:\Program Files\Rockwell Software\Arena™).
3. Select the over head person view picture from the library's list of pictures, see Figure 4.40. Then, select the Copy button. This will cause a copy of the picture to be made in the library.
4. After copying the picture, double-click on the picture. This will put you in the resource editor. This is like a drawing editor. For simplicity, just select the picture and change the fill color to green. If you want to make it look just like the picture in the text, you need to ungroup the picture elements and change their appearance as necessary.
5. Assign the white over head person view picture to the idle state and the green picture that was just made to the busy state. To change the idle picture, click the Idle button in the table on the left. Select from the picture library table on the right the picture of the white over head view person picture. Click the Transfer button between the tables

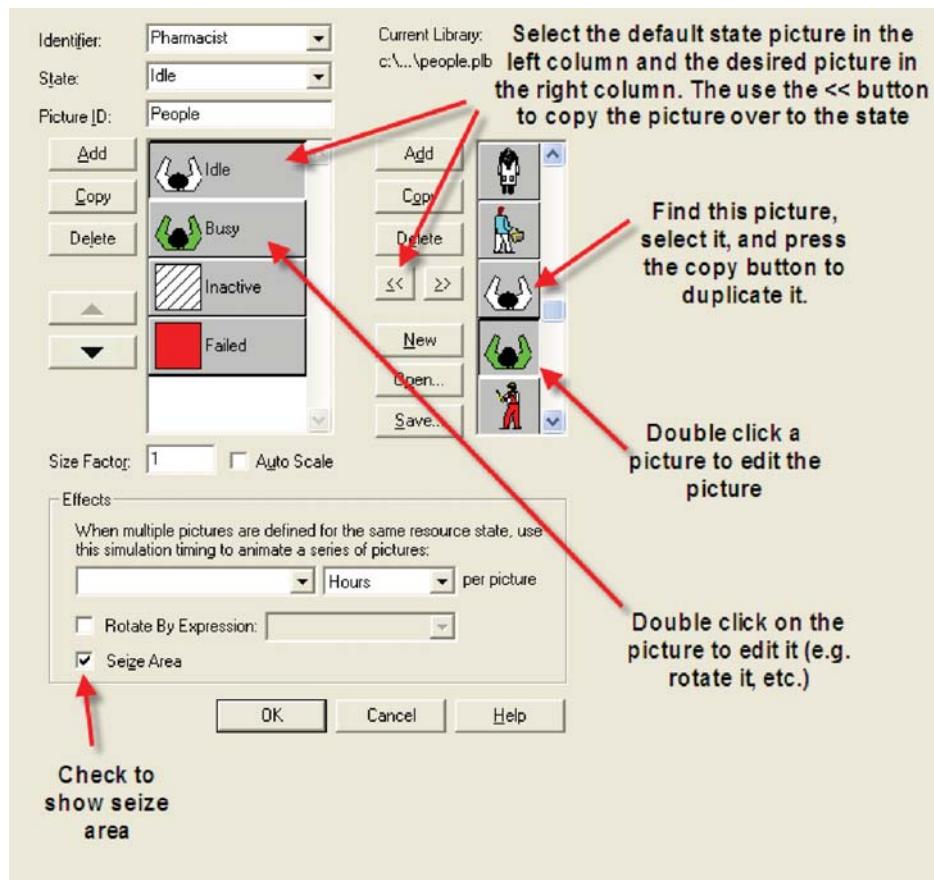
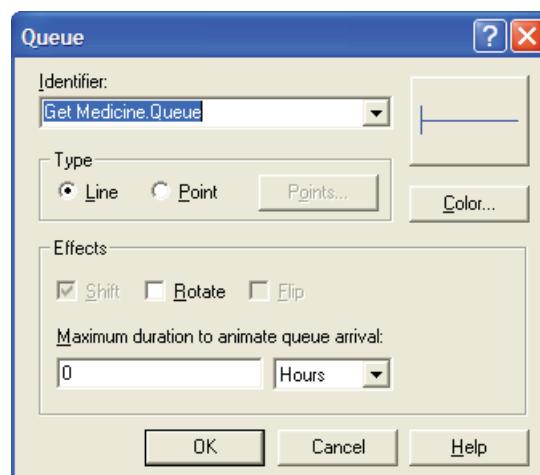


Figure 4.40 Resource animation state picture assignment.

to use the picture for the Idle resource state. To change the busy picture, click the Busy button in the table on the left. Select from the picture library table on the right the picture of green worker that was just made. Click the Transfer button between the tables to use the selected picture when the pharmacist is busy.

6. Now you need to rotate the picture so that the person is facing down. Double-click on the new Idle resource picture to open the resource editor. Then, you should select the whole picture and choose Arrange > Rotate in order to rotate the picture so that the pharmacist is facing down. Close the editor and do this for the Busy state also.
7. Click on the Seize Area check box. This will be discussed shortly.
8. Click OK to close the dialog box. (All other fields can be left with their default values.) Arena™ will ask about whether you want to save the changes to picture library. If you want to keep the green over head view person for use in later models, answer yes during the saving process. The cursor will appear as a cross-hair. Move it to the model window and click to place the pharmacist resource animation picture within the pharmacy.

Now if the pharmacist is placed in the pharmacy, it would be nice to show the pharmacist serving the current customer. This can be done by using the seize area for a resource. The seize area animates the entities that have seized the resource. By default, the seize area is not shown on the animation of a resource. The checking of the Seize Area check box allows the seize area to be visible. After placing the resource, zoom in on the resource and notice a small double circle. This is the seize area. Select the seize area and drag it so that it is positioned in the road adjacent to the pharmacy. Now, you need to place the queue of waiting cars on the road. Select the animation queue on top of the Get Medicine PROCESS module and delete it. Now, go to the Queue button on the Animate toolbar and select it. This will allow you to create a new queue animation element that is not physically attached to the Get Medicine PROCESS module and place it in the road. Fill in the Queue animation dialog as shown in Figure 4.41. Once you press OK, the cursor will turn in to a cross-hair and allow you to place the animation queue.

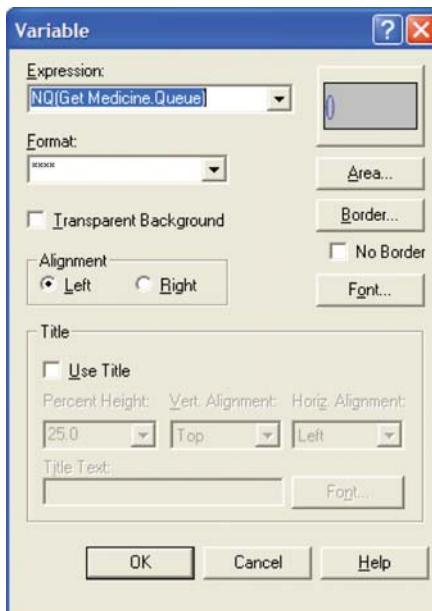


**Figure 4.41** Queue animation dialog.

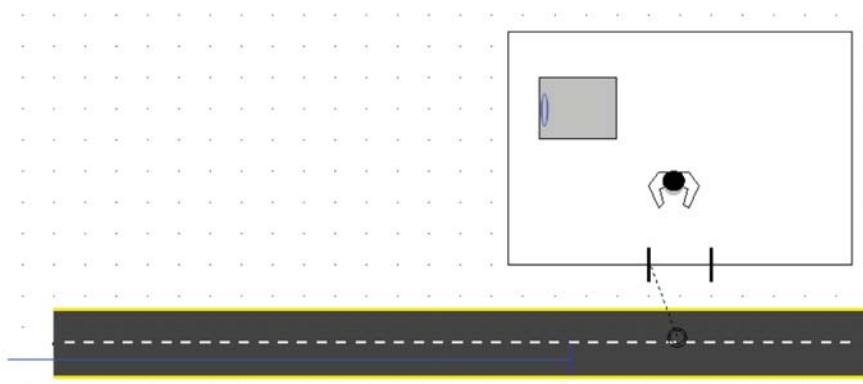
The final piece of animation that you will perform is to show the current number of waiting cars within the pharmacy. Select the Variable button on the Animate toolbar and fill out the resulting dialog as per Figure 4.42. The cursor should turn to cross-hairs and you should place the variable animation inside the pharmacy as shown in Figure 4.43.

The last thing to do before running the animation will be to turn off the animation associated with the flow chart connectors. This will stop the animation of the customers within the flow chart modules and be less of a distraction. The Object menu contains the option to disable/enable the animation of the connectors as shown in Figure 4.44.

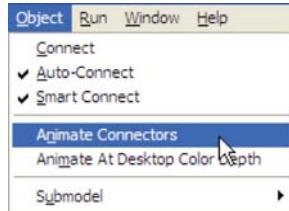
The final version of the animated pharmacy model is available in the file *PharmacyModule1WithAnimationS2.doe*. If you run the model with the animation on, you will see the



**Figure 4.42** Variable animation dialog.



**Figure 4.43** Final animation for pharmacy example.

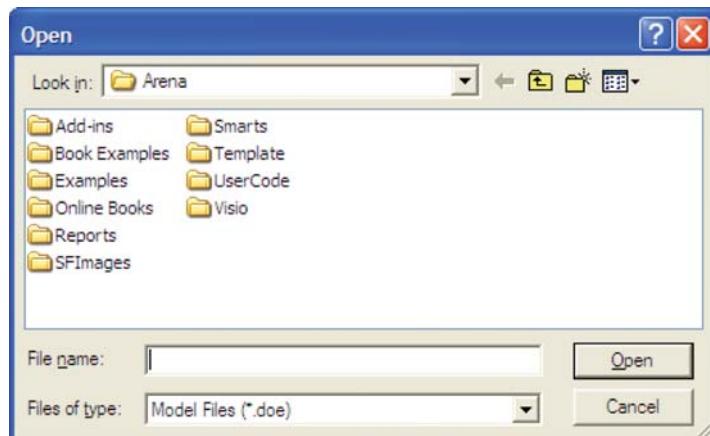


**Figure 4.44** Disabling connector animation.

pharmacist indicated as busy or idle, the car currently in service, any cars lined up waiting, and the current number of cars waiting as part of the animation.

Animation is an important part of the simulation process. It is extremely useful for showing to decision makers and experts to explain and ensure that the model is accepted. With this example, you have actually learned a great deal about animating an Arena™ model. One of the standard practices is to rely on the default animation that is available with the flow chart modules. Then, when animation is important for validation and for convincing decision makers, the simulation analyst can devote time to making the animation more worthy. Often, the animation is developed in one area of the model that can be easily accessed via the Navigate bar. Many detailed examples of what is possible with Arena™ animation come as demo models with the Arena™ distribution. For example, you should explore the models in the Arena™ Examples folder (e.g., *FlexibleManufacturing.doe*) (Figure 4.45). If you find that the Arena™ drawing tools are limiting, then you can use programs such as Visio or AutoCAD to make the drawings and import them as background for your Arena™ model. For more information on this process, search on importing Visio drawings or importing AutoCAD drawings within Arena's help system. Arena™ also has the capability for developing 3D animation; however, this is not discussed in this text.

The first part of this text primarily uses the default animation that comes with the placement of flow chart modules and concentrates on the analysis of simulation models; however, Chapter 9 returns to more on the topic of animation, when material handling constructs (e.g., conveyors and transporters) are examined.



**Figure 4.45** Arena folder structure.

## 4.8 GETTING HELP IN ARENA

Arena has an extensive help system. Each module's dialog box has a Help button, which will bring up help specific to that module. The help files use hyperlinks to important information. In fact, as can be seen in Figure 4.46, Arena<sup>TM</sup> has an overview of the modeling process as part of the help system.

The example files are especially useful for getting a feel for what is possible with Arena<sup>TM</sup>. This text will use a number of Arena's example models to illustrate various concepts. For example, the Smarts file folder has small Arena models that illustrate the use of particular modules. You are encouraged to explore and study the Arena<sup>TM</sup> Help system. In addition to the online help system, the Arena<sup>TM</sup> environment comes with user manuals in the form of PDF documents, which includes an introductory tutorial in the "Arena User's Guide" within the Online Books folder.

## 4.9 SIMAN AND THE RUN CONTROLLER

In a programming language such as Visual Basic or Java, programmers must first define the data types that they will use (int, float, double, etc.) and declare the variables, arrays, classes, etc. for those data types to be used in the program. The programmer then uses flow of control (if-then, while, etc.) statements to use the data types to perform the desired

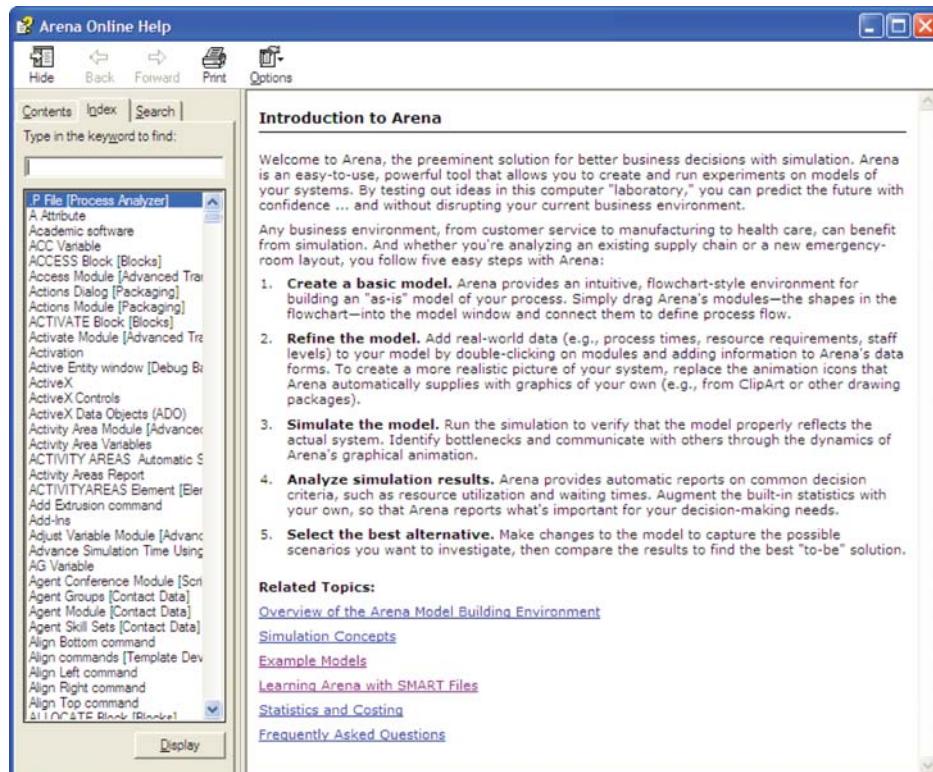


Figure 4.46 Arena<sup>TM</sup> help system.



**Figure 4.47** Run SIMAN View menu.

task. Programming in Arena™ is quite different from programming in a base language like C. An Arena™ simulation program consists of the flow chart and data modules that have been used or defined within the model. By dragging and dropping flow chart modules and filling in the appropriate dialog box entries, you are writing a program. Along with programming comes debugging and making sure that the program is working as it was intended. To be able to better debug your Arena™ programs, you must have, at the very least, a rudimentary understanding of Arena's underlying language called SIMAN. SIMAN code is produced by the Arena™ environment, compiled, and then executed. This section provides an overview of SIMAN and Arena's debugger called the Run Controller. To better understand the programming aspects of Arena™, it is useful to first dissect an Arena™ program.

#### 4.9.1 SIMAN MOD and EXP Files

The section reexamines the pharmacy model. Using the Run/SIMAN/View menu, generate two files associated with the SIMAN program, the *mod* (model) file and the *exp* (experiment) file (Figure 4.47). The files can be viewed using the Window menu within the Arena™ environment.

The *mod* file contains the SIMAN representation for the flow chart modules that were laid out in the model window. The *exp* file contains the SIMAN representation for the data modules and simulation run control parameters that are used during the execution of the simulation. *mod* stands for model, and *exp* stands for experiment.

Listing 4.1 presents the experiment (*exp*) file for the pharmacy model. For readers who are familiar with C or C++ programming, the *exp* file is similar to the header files used in those languages. The experiment file defines the elements that are to be used by the model during the execution of the simulation. As indicated for this file, the major elements are categorized as PROJECT, VARIABLES, QUEUES, RESOURCES, PICTURES, REPLICATE, and ENTITIES. Additional categories that are not used (and thus not shown) in this model include TALLIES and OUTPUTS. Each of these constructs will be discussed as you learn more about the modules within the Arena™ environment. The experiment module declares and defines certain elements that will be used in the model. For example, the RESOURCES element indicates that there is a resource called Pharmacist that can be

**Listing 4.1 SIMAN exp File Contents for Pharmacy Model**

```

PROJECT,      "Unnamed Project", " ", , , No, Yes, Yes, Yes, No, No, No,
No, No;

VARIABLES:    Get Medicine.WIP,CLEAR(System),CATEGORY("Exclude-
              Exclude"),DATATYPE(Real):
              Dispose 1.NumberOut,CLEAR(Statistics),
              CATEGORY("Exclude"):
              Create Customers.NumberOut,CLEAR(Statistics),
              CATEGORY("Exclude"):
              Get Medicine.NumberIn,CLEAR(Statistics),
              CATEGORY("Exclude"):
              Get Medicine.NumberOut,CLEAR(Statistics),
              CATEGORY("Exclude");

QUEUES:       Get Medicine.Queue,FIFO,,AUTOSTATS(Yes,,);

PICTURES:     Picture.Airplane:
              Picture.Green Ball:
              Picture.Blue Page:
              Picture.Telephone:
              Picture.Blue Ball:
              Picture.Yellow Page:
              Picture.EMail:
              Picture.Yellow Ball:
              Picture.Bike:
              Picture.Report:
              Picture.Van:
              Picture.Widgets:
              Picture.Envelope:
              Picture.Fax:
              Picture.Truck:
              Picture.Person:
              Picture.Letter:
              Picture.Box:
              Picture.Woman:
              Picture.Package:
              Picture.Man:
              Picture.Diskette:
              Picture.Boat:
              Picture.Red Page:
              Picture.Ball:
              Picture.Green Page:
              Picture.Red Ball;

RESOURCES:    Pharmacist,Capacity(1),,COST(0.0,0.0,0.0),
              CATEGORY(Resources),,AUTOSTATS(Yes,,);

REPLICATE,    1,,HoursToBaseTime(10000),Yes,Yes,,,24,Minutes,
              No,No,,,Yes;

ENTITIES:     Customer,Picture.Van,0.0,0.0,0.0,0.0,0.0,0.0,
              AUTOSTATS(Yes,,);

```

**Listing 4.2 SIMAN mod File Contents for Pharmacist Model**

```

;
;      Model statements for module:  BasicProcess.Create 1
;      (Create Customers)
;
2$          CREATE,           1,MinutesToBaseTime(expo(6)),
Customer:MinutesToBaseTime(EXPO(6)):NEXT(3$);

3$          ASSIGN:      Create Customers.NumberOut=Create
Customers.NumberOut + 1:NEXT(0$);
;

;
;      Model statements for module:  BasicProcess.Process 1
;      (Get Medicine)
;
0$          ASSIGN:      Get Medicine.NumberIn=Get Medicine.
NumberIn + 1:
                                Get Medicine.WIP=Get Medicine.WIP+1;
9$          QUEUE,
                                Get Medicine.Queue;
8$          SEIZE,
                                2,VA:
                                Pharmacist,1:NEXT(7$);

7$          DELAY:      expo(3),,VA;
6$          RELEASE:    Pharmacist,1;
54$          ASSIGN:     Get Medicine.NumberOut=Get Medicine.
NumberOut + 1:
                                Get Medicine.WIP=Get Medicine.WIP-1:
                                NEXT(1$);
;
;
;      Model statements for module:  BasicProcess.Dispose 1
;      (Dispose 1)
;
1$          ASSIGN:     Dispose 1.NumberOut=Dispose 1.
NumberOut + 1;
57$          DISPOSE:    Yes;

```

used in the model. Understanding the exact syntax of the SIMAN elements is not necessarily important, but being able to look for and interpret these elements can be useful in diagnosing certain errors that occur during the model debugging process. It is important to understand that these elements are defined during the model building process, especially when data modules are filled out within the Arena™ environment.

Listing 4.2 presents the contents of the SIMAN model (mod) file for the Pharmacy model. The model file is similar in spirit to the “\*.c” or “\*.cpp” files in C/C++ programming. It represents the flow chart portion of the model. In this code, “;” indicates a comment line. The capitalized commands represent special SIMAN language keywords. For example, the ASSIGN keyword is used to indicate an assignment statement, that is, *variable = expression*. There are two keywords that you should immediately recognize from the model window, CREATE and DISPOSE. Each noncommented line has a line number that

identifies the SIMAN statement. For example, 57\$ identifies the DISPOSE statement. These line numbers are especially useful in following the execution of the code. For example, within the code, you should notice the use of the NEXT() keyword. For example, on line number 54\$, the keyword NEXT(1\$) redirects the flow of control to the line statement 1\$. You should also notice that many lines of code are generated from the placement of the three modules (CREATE, PROCESS, DISPOSE). Much of this SIMAN code is added to the model to enable additional statistical collection. The exact syntax associated with this generated SIMAN code will not be the focus of the discussion; however, the ability to review this code and interpret its basic functionality is essential when developing and debugging complex models. The code generated within the experiment and model files is not directly editable. The only way to change this code is to change the model in the data modules or within the model window. Finally, the SIMAN code is not directly executed on the computer. Before running the model, the SIMAN code is checked for syntax or other errors and then is translated to an executable form via a C/C++ compiler translation.

In a programming language like “C,” the program starts at a clearly defined starting point, for example, the “main()” function. Arena™ has no such main() function. Examining the actual SIMAN code that is generated from the model building process does not yield a specific starting point either. So where does Arena™ start executing? To fully answer this question involves many aspects of simulation that have not yet been examined. Thus, at this point, a simplified answer to the question will be presented.

A simulation program models the changes in a system at specific events in time. These events are scheduled and executed by the simulation executive ordered by time. Arena™ starts executing with the first scheduled event and then processes each event sequentially until there are no more events to be executed. Of course, this begs the question, how does the first event get scheduled? In Arena™, the first event is determined by the CREATE module with the smallest “First Creation” time. Figure 4.48 illustrates the dialog box for the CREATE module. The text field labeled “First Creation” accepts an expression that must evaluate to a real number. In the figure, the expo() function is used to generate a random variable from the exponential distribution with a mean of 6 min to be used as the time of the event associated with the creation of an entity from this CREATE module.

The careful reader should have the natural question, “What if there are ties?” In other words, what if there are two or more CREATE modules with a “First Creation” time set to the same value. For example, it is quite common to have multiple CREATE modules with

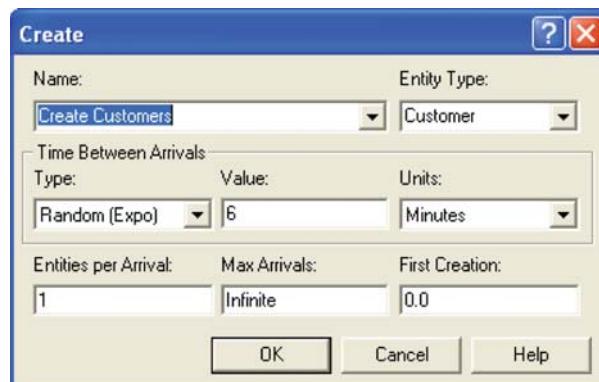
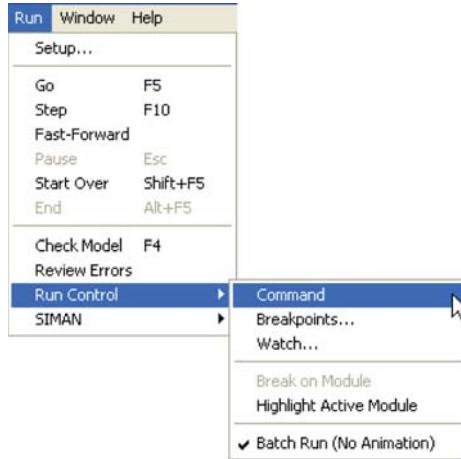


Figure 4.48 CREATE module dialog Box.



**Figure 4.49** Invoking the Run Controller.

their *First Creation* time set at zero. Which CREATE module will go first? This is where looking at the SIMAN model output is useful. The CREATE module with the smallest *First Creation* time that is listed first in the SIMAN model file will be the first to execute. Thus, if there are multiple CREATE modules each with a *First Creation* time set to zero, then the CREATE module that is listed first will create its entity first.<sup>1</sup>

#### 4.9.2 Using the Run Controller

An integral part of any programming effort is debugging the program. Simulation program development in Arena™ is no different. You have already seen how to do some rudimentary tracing via the READWRITE module; however, this section discusses some of the debugging and tracing capabilities that are built into the Arena™ environment. Arena's debugging and tracing facilities come in two major forms: the run controller and animation. Some of the basic capabilities of Arena's animation have already been demonstrated. Using animation to assist in checking your model logic is highly recommended. Aspects of animation will be discussed throughout this text; however, this section concentrates on using Arena's Run Controller. Arena's Run Controller is found on the Run menu as shown in Figure 4.49.

Figure 4.50 illustrates how the Arena™ environment appears after the Run Controller has been invoked on the original pharmacy model. The Run Controller allows tracing and debugging actions to be performed by the user through the use of various commands. Let us first learn about the various commands and capabilities of the Run Controller. Go to Arena™ Help, and in the index search, type in “Command-Driven Run Controller Introduction.” You should see a screen that looks something like that shown in Figure 4.51.

The Run Controller allows the Arena™ system to be executed in a debugging mode. With the Run Controller, you can set trace options, watch variables and attributes, run the model until certain time or conditions are true, examine the values of entities attributes, etc. Within the help system, use the hyperlink that says “Run Controller Commands.” You

<sup>1</sup>Again, this is a simplified discussion. There are other ways to get an initial event scheduled and Arena™ uses some internally generated events.

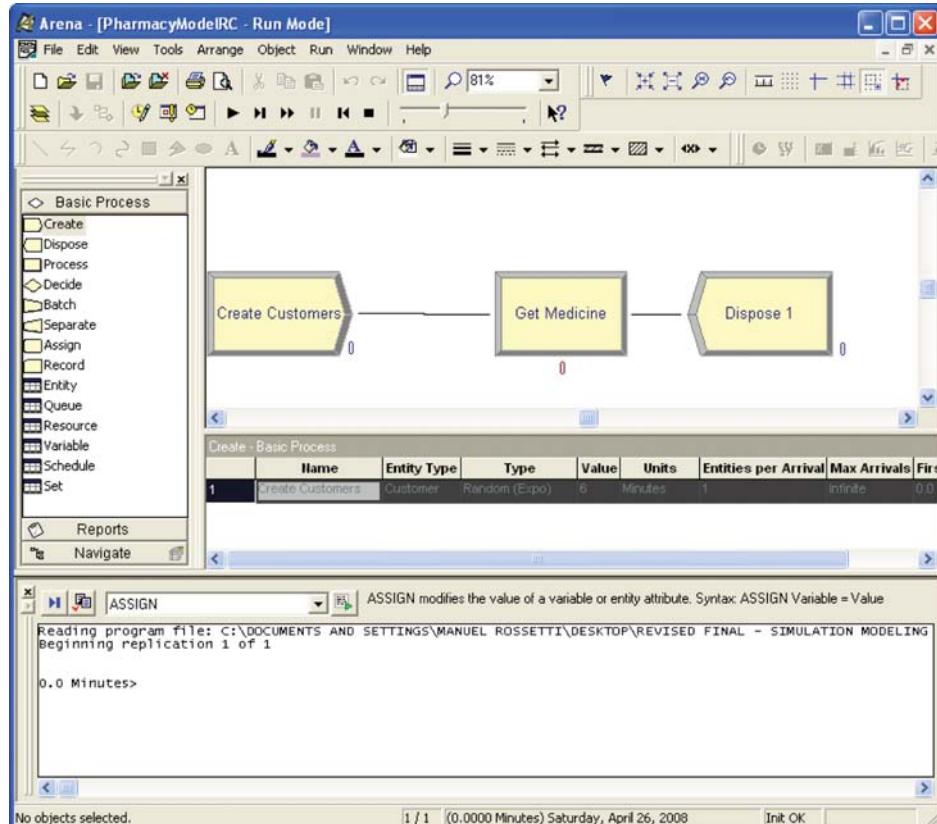


Figure 4.50 Arena<sup>TM</sup> environment with Run Controller invoked.

will see a list of commands for the Run Controller. Selecting any of these provides detailed instructions on how to use the command within the Run Controller environment.

In what follows, some simple commands will be used to examine the drive-through pharmacy model. Open the pharmacy model within Arena<sup>TM</sup> and then open up the CREATE module and make sure that the time of the first arrival is set to 0.0. In addition, open up the PROCESS module and make sure that the delay expression is EXPO(5). This will help in ensuring that your output looks like what is shown in the figures. The first arrival will be discussed in what follows.

Now, you should invoke the Run Controller as indicated in Figure 4.49. This will start the run of the model. Arena<sup>TM</sup> will immediately pause the execution of the model at the first execution step. You should see a window as illustrated in Figure 4.50. Let us first turn the tracing on for the model. By default, Arena<sup>TM</sup> embeds tracing output into the SIMAN code that is generated. Information about the current module and the active entity will be displayed (Figures 4.52 and 4.53). To turn on the tracing, enter “set trace” at the prompt or select the command from the drop down menu of commands, then press return to enter the command. You can also press the toggle trace button.

Now select the VCR-like run button for single stepping through the model. The button is indicated in Figure 4.54. You should press the single step button again to execute another step of the model. Your command window should look like Figure 4.54. Remember that

### Command-Driven Run Controller Introduction

A model can compile without errors, but may still produce invalid results when executed. Debugging is the process of isolating and correcting the errors that produce invalid results. The Run Controller allows you to monitor interactively the execution of the simulation so that errors can be isolated and corrected.

The key to debugging is to determine what is happening at critical points in the model. The Run Controller allows you to step through or suspend execution of the model at critical points (called breakpoints) to examine the values of system status variables.

The Run Controller is invoked by selecting the [Command](#) option from the Run menu. It can be invoked at the start of the simulation run or at any time during execution by first pressing the Escape key. The Run Controller then prompts you to enter commands interactively from the keyboard.

See [Run Controller Commands](#) for a complete listing of available commands.

The command prompt is marked by the current value of simulated time, TNOW. Execute commands by typing the command name and any modifying keywords and operands at the prompt, and then hitting Enter.

#### Some Rules to Remember:

- Two- or three-letter abbreviations are sufficient to specify any command or keyword uniquely when entering a command.
- The Escape key may be used to abort the display generated by a given command.
- Unique identification numbers are assigned to each entity. These numbers appear in trace statements as entities move through the model, and when using certain commands to view queue contents, the event calendar, and so on.
- Intercepts, traces, and watch points set on entities apply only to the active entity—the entity currently executing blocks. Attempting to set an intercept, trace, or watch point when the selected entity does not currently exist in the model will result in an error. Setting a break point ensures that an entity is active when the break is reached; see the [SET BREAK command](#).
- To end the current simulation replication with a summary report, use the [END](#) command. The [QUIT](#) command terminates the simulation run without issuing a summary report.
- The Run Controller assigns identification numbers when setting watch points or trace output based on conditions, expressions, or times. When canceling these watch points or trace conditions, the identifier may be used instead of the exact condition, expression, or time.

In some cases, repeats of an operand or group of operands may be specified by separating each set of repeated operands with a comma. Also in some cases, a range of values may be entered. When indicating a range, use a hyphen (-) or double periods (...) to separate the low and high numbers, or use the wild card symbol (\*) to represent all values. When entering expressions, spaces are treated as punctuation or part of symbol names so extra spaces should not be included.

**Figure 4.51** Arena™ Help on Run Controller.

the CREATE module was set to create a single arrival at time 0.0. This is exactly what is displayed. In addition, the trace output is telling you that the next arrival will be coming at time 2.0769136. The asterisk marking the output indicates the SIMAN statement that will be executed on the *next* step. Then, when the step executes, the trace results are given and the next statement to execute is indicated.

If you press the single step button twice, you will see that ASSIGN statements are executed and that the current entity is about to execute a QUEUE statement. After seven steps, the run controller output should appear as shown in Figure 4.55.

From this trace output, it is apparent that the entity was sent directly to the SEIZE block from the QUEUE block where it seized one unit of the pharmacist. Then, the entity proceeded to the DELAY block where it was delayed by 2.2261054 time units. This is the service time of the entity. Since the entity is entering service at time 0.0, it will thus leave at time 2.2261054. In summary, Entity 2 was created at time 0, arrived to the system, and

<b>Run Controller Commands</b>	
The following Run Controller Commands are available:	
<b>General Commands</b>	
<a href="#">ASSIGN</a>	<a href="#">CLEAR</a>
<a href="#">EVENT</a>	<a href="#">GO</a>
<a href="#">QUIT</a>	<a href="#">SIGNAL</a>
<a href="#">SHOW</a>	
<b>Cancel Commands</b>	
<a href="#">CANCEL BREAK</a>	<a href="#">CANCEL INTERCEPT</a>
<a href="#">CANCEL TRACE BLOCKS</a>	<a href="#">CANCEL TRACE CONDITIONS</a>
<a href="#">CANCEL TRACE ENTITIES</a>	<a href="#">CANCEL TRACE EXPRESSIONS</a>
<a href="#">CANCEL TRACE FILE</a>	<a href="#">CANCEL TRACE TIMES</a>
<a href="#">CANCEL WATCH</a>	
<b>Set Commands</b>	
<a href="#">SET BREAK</a>	<a href="#">SET INTERCEPT</a>
<a href="#">SET MODEL</a>	<a href="#">SET TRACE</a>
<a href="#">SET TRACE BLOCKS</a>	<a href="#">SET TRACE CONDITIONS</a>
<a href="#">SET TRACE ENTITIES</a>	<a href="#">SET TRACE EXPRESSIONS</a>
<a href="#">SET TRACE FILE</a>	<a href="#">SET TRACE TIMES</a>
<a href="#">SET WATCH</a>	
<b>View Commands</b>	
<a href="#">VIEW</a>	<a href="#">VIEW BREAK</a>
<a href="#">VIEW CALENDAR</a>	<a href="#">VIEW CONVEYORS</a>
<a href="#">VIEW ENTITY</a>	<a href="#">VIEW INTERCEPT</a>
<a href="#">VIEW MODEL</a>	<a href="#">VIEW QUEUE</a>
<a href="#">VIEW SOURCE</a>	<a href="#">VIEW TRACE</a>
<a href="#">VIEW WATCH</a>	

Figure 4.52 Arena™ Run Controller commands.

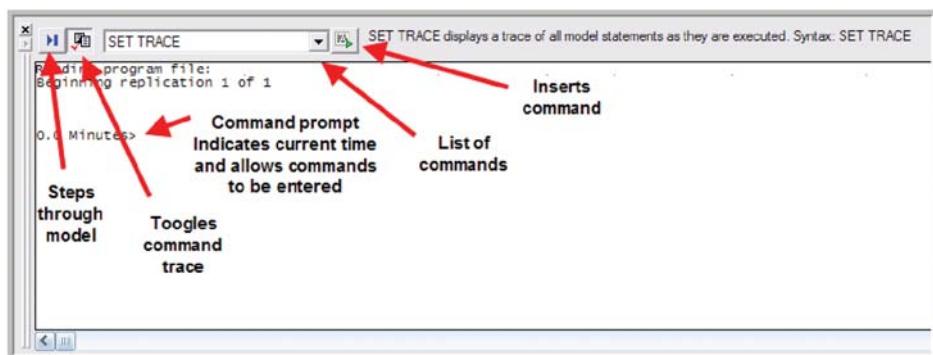


Figure 4.53 Using commands in the Run Controller.

```

SET TRACE
Reading program file:
Beginning replication 1 of 1

0.0 Minutes>STEP
SIMAN System Trace Beginning at Time: 0.0 ← Results from first step
Seq# Label Block System Status Change
Time: 0 Entity: 2
SIMAN Run Controller. ← Indicates the next command to be executed
* 1 2$ CREATE,1,MinutesToBaseTime(0.0),Customer:
MinutesToBaseTime(EXPO(6));
NEXT(3$);

0.0 Minutes>STEP
1 2$ CREATE ← Results from second step
Entity Type set to Customer
Next creation scheduled at time 2.0769136
Batch of 1 Customer entities created
* 2 3$ ASSIGN:
Create Customers.NumberOut=Create Customers.NumberOut
t+1;
NEXT(0$);

0.0 Minutes>

```

Figure 4.54 Run Controller output window after two steps.

```

SET TRACE
QUEUE Entity 2 sent to next block
* 5 8$ SEIZE,2,VA:Pharmacist,1:NEXT(7$);

0.0 Minutes>STEP
5 8$ SEIZE Tally Get Medicine.Queue.WaitingTime recorded 0.0
Seized 1.0 unit(s) of resource Pharmacist
* 6 7$ DELAY:expo(5),,VA;

0.0 Minutes>STEP
6 7$ DELAY Delayed by 2.2261054 until time 2.2261054
Time: 2.0769136 Entity: 3
SIMAN Run Controller.
* 1 2$ CREATE,1,MinutesToBaseTime(0.0),Customer:
MinutesToBaseTime(EXPO(6));
NEXT(3$);

2.0769136 Minutes>

```

Figure 4.55 Run Controller output window after seven steps.

attempted to get a unit of the pharmacist. Since the pharmacist was idle at the start of the simulation, the entity did not have to wait in queue and was able to immediately start service.

Now, something new has happened. The time has jumped to 2.0769136. As you might recall, this was the time indicated by the CREATE module for the next entity to arrive. This entity is denoted by the number 3. Entity 3 is now the active entity. What happened to Entity 2? It was placed on the future events list (FEL) by the DELAY module and is scheduled to “wake up” at time 2.2261054. By using the VIEW CALENDAR command,

```

*** Entities on current events chain: 0 ***

*** Entities on future events heap : 2 ***

*** Entity 2 to arrive at time 2.2261054 at block
    7 6$          RELEASE:Pharmacist,1;

Entity.SerialNumber = 1
Entity.Type = 1
Entity.Picture = 11
Entity.Station (M) = 0
Entity.Sequence (NS) = 0
Entity.Jobstep (IS) = 0
Entity.CurrentStation = 0
Entity.PlannedStation = 0
Entity.CreateTime = 0.0
Entity.StartTime = 0.0
Entity.NVATime = 0.0
Entity.WaitTime = 0.0
Entity.TranTime = 0.0
Entity.OtherTime = 0.0

*** Entity 1 to cause end of replication at time 600000.0

2.0769136 Minutes>

```

**Figure 4.56** VIEW CALENDAR OUTPUT.

you can see all the events that are scheduled on the event calendar. Figure 4.56 illustrates the output of the VIEW CALENDAR command. There are two data structures within Arena™ to hold entities that are scheduled. The “current events chain” represents the entities that are scheduled to occur at the current time. The “future events heap” represents those entities that are scheduled in the future.

This output indicates that no other entities are scheduled for the current time and that Entity 2 is scheduled to activate at time 2.2261054. The entity is not going to “arrive” as indicated. This is a typo in the Arena™ output. The actual word should be “activate.” In addition, the calendar indicates that an entity numbered 1 is scheduled to cause the end of the simulation at time 600000.0. Arena™ uses an internal entity to schedule the event that represents the end of a replication. If there are a large number of entities on the calendar, then the VIEW CALENDAR command can be very verbose.

Single stepping the model allows the next entity to be created and arrive to the model. This was foretold in the last trace of the CREATE module.

The run controller provides the ability to stop the execution when a condition occurs. Suppose that you are interested in examining the attributes of the customers in the queue when at least two are waiting. You can set a watch on the number of customers in the queue, and this will cause Arena™ to stop whenever this condition becomes true. Then, by using the VIEW QUEUE command, you can view the contents of the customer waiting queue. Carefully type in SET WATCH NQ(Get Medicine.Queue) == 2 at the run controller prompt and hit return. Your run controller should look like Figure 4.57.

If you type GO at the run controller prompt, Arena™ will run until the watch condition is met, see Figure 4.58. The VIEW QUEUE command will show all the entities in all the queues, see Figure 4.59. You can also select only certain queues and only certain entities. See the Arena’s help for how to specialize the commands.

Arena™ also has a debugging bar within the run command controller that can be accessed from the Run > Run Control > Breakpoints menu, see Figure 4.49. This allows the user to easily set breakpoints, view the calendar, view the attributes of the active entity, and set watches on the model. The debug bar is illustrated for this execution after the next step in Figure 4.60. In the figure, the active entity tab is open to allow the attributes of the

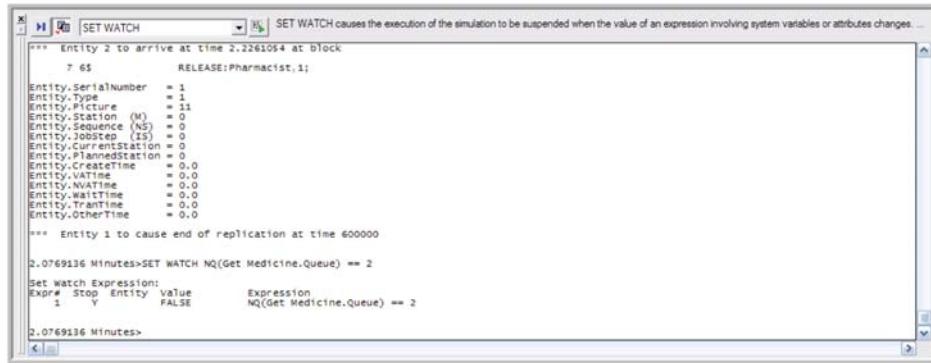


Figure 4.57 Setting a WATCH.

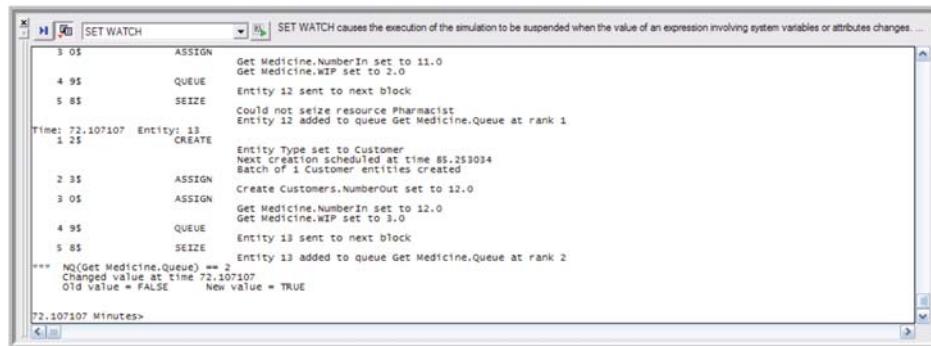


Figure 4.58 Output after WATCH condition break.

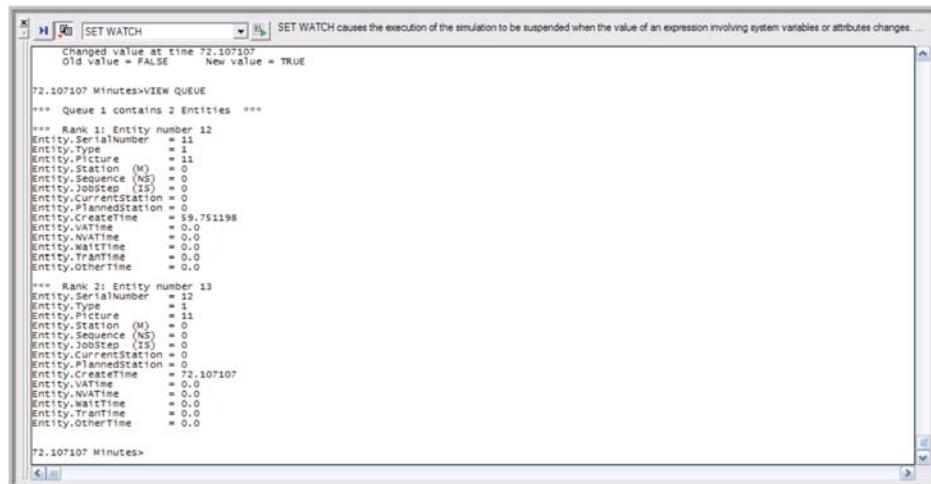
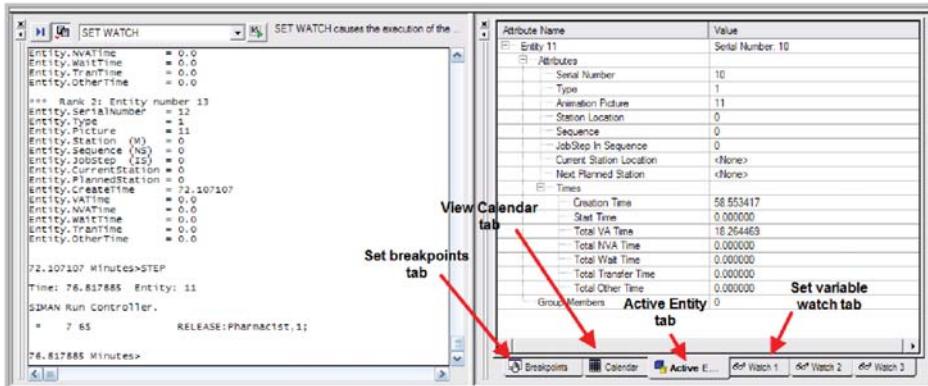


Figure 4.59 Output after VIEW QUEUE command.



**Figure 4.60** Debug window within Run Controller.

entity to be seen. The functionality of the debug bar is similar to many debuggers. The Arena™ help system has a discussion on how to use the tabs within the debug bar. You can find out more by searching for *Debug Bar* in Arena's help system.

There are many commands that you can try to use during your debugging. Only a few commands have been discussed here. What do you think the following commands do?

- go until 10.0
- show NR(\*)
- show NQ(\*)
- view entity

Restart your model. Turn on the tracing, type them in, and find out!

There are just a couple of final notes to mention about the run controller. When the model runs to completion or if you stop it during the run, the run controller window will show the text-based statistical results for your model, which can be quite useful. In addition, after you are done tracing your model, be sure to turn the tracing off (CANCEL TRACE) and to save the model with the tracing off. After you turn the tracing on for a model, Arena™ remembers this even though you might not be running the model via the run controller. The trace output will still be produced and will significantly slow the execution of your model down. In addition, the memory requirements of the trace output can be quite large and you may get out of memory errors for what appears to be inexplicable reasons. Do not forget to turn the tracing off and to resave!

## 4.10 HOW ARENA MANAGES ENTITIES AND EVENTS

By using the run controller, you can start to get a feel for how Arena™ works. That is, how it processes the entities. The run controller facilitates the tracking and tracing of entities as they move through the system. This section provides a conceptual model for how Arena™ processes the entities. The discussion in this section is based in large part on the discussion found in Schriber and Brunner [1998].

The processing of entities within Arena™ is based on a process-oriented view. The modeler describes the life of an entity as it moves through the system. In some sense, the system

processes the entities. The entities enter the system (via CREATE modules), get processed (via various flow chart modules), and then depart the system (via DISPOSE modules). The entities act as transactions or units of work that must be processed. As the entities move through the processing, they cause events that change the state of the system to occur. In addition, the entities cause future events to be scheduled.

There is a direct mapping of entity movement to event processing. Simulation languages have a construct called the event calendar that holds the events that are scheduled to occur in the future. This is similar to your own personal work calendar. Future events are placed on the calendar and as time passes, the event's alarm goes off and then the event is processed. In discrete-event modeling, the clock gets updated only when an event happens, causing the system to move through time. The movement of entities cause the events to be scheduled and processed.

According to Schriber and Brunner [1998], there are two phases for processing events: the entity movement phase (EMP) and the clock update phase (CUP). The CUP is straightforward: when all events have been processed at the current time, update the simulation time to the time of the next scheduled event and process any events at this new updated time. The processing of an event corresponds to the EMP. Thus, a key to understand how this works is to understand how entities move within a model. One thing to keep in mind throughout this discussion is that only one event can be processed at a time. That is, only one entity can be moving at a time. It may seem obvious that if only one entity can be moving at a time, then all the other entities are not moving. However, the entities that are not moving have different reasons, given their current state.

Schriber and Brunner [1998] divided the life of an entity into five states:

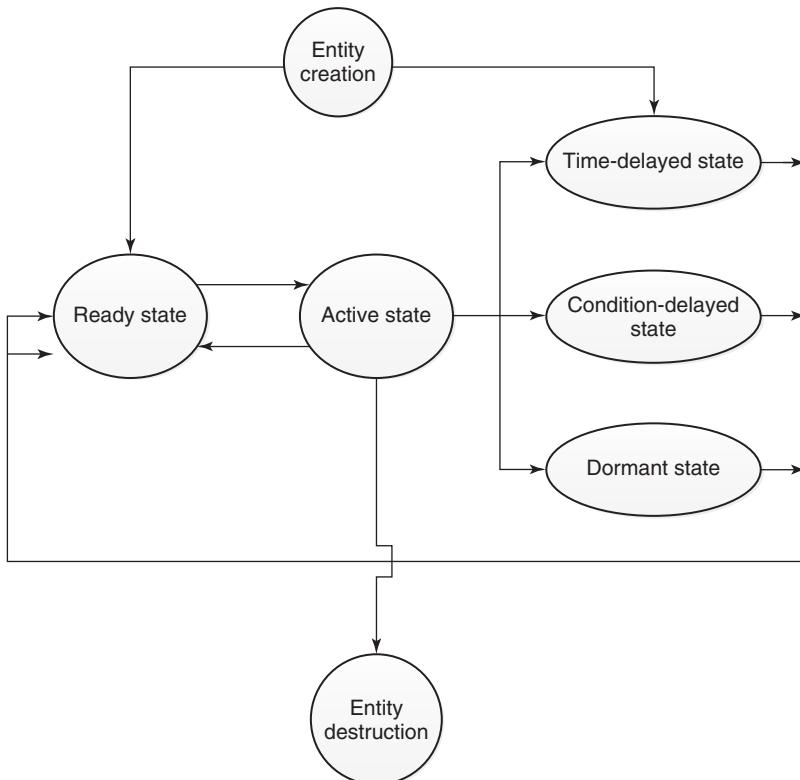
1. *Active State.* The active state represents the entity that is currently being processed. The entity that is currently moving is in the active state. We call the entity in the active state, the *active entity*. To use a rugby analogy, the active entity is the one with the ball. The active entity keeps running through the model (with the ball) until it encounters some kind of delay. The active entity then gives up the ball to another entity and transitions into an alternative state. The new active entity (with the ball) then starts moving through the model.
2. *Ready State.* Entities in the ready state are ready to move at the current simulation time. In the rugby analogy, they are ready to receive the ball so that they can start running through the model. There can be more than one entity in the ready state. Thus, a key issue that must be understood is how to determine which of the ready entities will receive the ball, that is, start moving next. Ready state entities transition into the active state.
3. *Time-Delayed State.* Entities in the time-delayed state are entities that are scheduled to transition into the ready state at some known future time. Time-delayed entities have an event scheduled in the event calendar. The purpose of the event is to cause the entity to transition into the ready state. When the active entity executes an operation that causes a delay, such as the delay within a PROCESS module, an event is scheduled and the active entity is placed in the time-delayed state.
4. *Condition-Delayed State.* Entities in the condition-delayed state are waiting for a specific condition within the model to become true so that they can move into the ready state. For example, an entity that is waiting for a unit of a resource to become available is in the condition-delayed state. When the condition causing the waiting is removed,

the EMP logic will evaluate whether or not the condition-delayed entity becomes a ready state entity. This processing will occur automatically.

5. *Dormant State*. The dormant state represents the situation in which an entity is not waiting on time or a condition that can be automatically processed within the model. The dormant state represents the situation in which specialized logic must be added to the model in order to cause the entity to transition from the dormant state to the ready state.

Figure 4.61 illustrates how an entity can transition between the five states. As illustrated in the figure, when an entity is created, it is placed in the ready state or the time-delayed state. CREATE modules place entities in the time-delayed state and the SEPARATE module<sup>2</sup> place entities in the ready state. An entity in the ready state can only become active. The active entity can transition into the ready state, any of the three delay states (time delayed, condition delayed, and dormant), or be destroyed (DISPOSE). Any of the three delay states (time delayed, condition delayed, and dormant) may transition to the ready state.

When an entity is not the active entity, it must be held somewhere in memory. Entities that are in the ready state are held in the current events list (CEL). The CEL holds all the



**Figure 4.61** Illustration of entity state transitions. Based on Figure 24.9 of Schriber and Brunner [1998].

<sup>2</sup>Discussed in Chapter 5.

entities that are candidates to start moving at the current time. The list is processed in first out manner. Entities can be placed on the CEL in four major ways:

1. The entity was a time-delayed entity and they are now scheduled to move at the current time.
2. The entity was a condition-delayed entity and the blocking condition has been removed.
3. The entity was in the dormant state and the user executed logic to move the entity to the CEL.
4. The entity executed a SEPARATE module that duplicates the active entity. The duplicates are automatically placed on the CEL and become ready state entities.

Entities that are in the time-delayed state are in the FEL. When an entity executes a module that schedules a future event, the entity is placed in the delayed state and held in the FEL. Entities in the condition-delayed state are held in a delay list (DL) that is tied to the condition. Delay lists map to queue constructs within the model. For example, the PROCESS module with the SEIZE, DELAY, RELEASE option automatically defines a QUEUE that holds the entities that are waiting for the resource. There can be many DLs within the model at any time. For example, the BATCH<sup>3</sup> module's queue is also a DL.

Finally, entities that are in the dormant state are held in user-managed lists (UML). In Arena™ these are queues that are not tied to a specific condition. HOLD<sup>4</sup> modules allow for an “infinite hold” option, which defines a UML. Thus, the entities within the model can be in a number of different states and be held in different lists as they are processed. Table 4.1 illustrates that many entities can be in the model at the same time in and each entity will be in a state and a list.

The EMP and CUP within Arena™ can be summarized with the following steps:

1. Remove entity at the top of the CEL and make it active.
2. Allow active entity to move through model.
  - If the active entity makes any duplicates, they are placed on the top of CEL. They will be last in first out among the ready state entities, but first in first out among themselves.
  - If the active entity removes any entities from user-defined lists, they will be placed last in first out order on the CEL.

**TABLE 4.1 Entity Records**

IDENT	Entity.SerialNumber	Size	Weight	Processing Time	State	List
1	1001	2	33	20	Active	
2	1002	3	22	55	Ready	CEL
3	1003	1	11	44	Time delayed	FEL
4	1001	2	33	20	Ready	CEL
5	1004	5	10	14	Condition delayed	DL
6	1005	4	14	10	Dormant	UML

<sup>3</sup>Discussed in Chapter 5.

<sup>4</sup>Discussed in Chapter 8.

3. When the active entity stops moving, the entities on the CEL are processed, the top entity is selected and made active and allowed to move. This repeats until there are no more active entities in the CEL.
4. When there are no more entities on the CEL, the EMP logic checks to see if any conditions have changed related to entities that are in the condition-delayed state. Any qualifying condition-delayed entities are transferred to the CEL, it is processed as previously described. When there are no more entities on the CEL and no qualifying condition-delayed entities, the CUP begins.
5. The CUP causes simulated time to be advanced to the next event on the FEL and initiates the processing of the FEL.
6. When processing the FEL, the logic removes any entities on the FEL that are scheduled to occur at the current time. The time-delayed entities are moved from the FEL to the CEL. If there is more than one entity scheduled to move at the current time, the entities are placed last in first out on the CEL. There is one caveat here. Arena™ may use an internal entity to effect logic within the model. An example internal entity is the entity that is scheduled to cause the simulation to end. Internal entities will be processed immediately, without being placed on the CEL.
7. When there are no more entities to process, the simulation will automatically stop.

This discussion should provide you with a basic understanding of how Arena™ processes entities. In addition, it should make your use of the run controller more effective. If you can better conceptualize what is happening to each entity, then you can better verify that your model is working as intended.

Why does understanding entity processing matter? There are some common modeling situations that happen automatically, for which you may need to understand the default processing logic within Arena™. Schriber and Brunner [1998] discussed three of these modeling situations. Here we will just mention one case. Suppose that a customer releases a resource and then immediately tries to seize the resource again. What happens? As outlined in Schriber and Brunner [1998], there are three logical possibilities:

1. Immediately upon the release, the resource is allocated to a waiting customer. Since the active entity is not a waiting customer, it cannot be allocated to the resource.
2. The allocation of the resource does not occur until the active entity stops moving. Thus, it would be a contender for the resource.
3. The releasing entity recaptures the resource immediately, without regard to any other waiting entities.

These three possibilities seem perfectly reasonable. What does Arena™ do? The first option is done automatically in Arena™. If this is not the behavior that you desire, then you may have to implement special logic.

## 4.11 SUMMARY

This chapter introduced the Arena™ environment. Using this environment, you can build simulation models using a drag and drop construction process. The Arena™ environment

facilitates the model building process, the model running process, and the output analysis process.

The modules covered included the following:

**CREATE** Used to create and introduce entities into the model according to a pattern.

**DISPOSE** Used to dispose of entities once they have completed their activities within the model.

**PROCESS** Used to allow an entity to experience an activity with the possible use of a resource.

**ASSIGN** Used to make assignments to variables and attributes within the model

**RECORD** Used to capture and tabulate statistics within the model.

**DECIDE** Used to provide alternative flow paths for an entity based on probabilistic or condition based branching.

**VARIABLE** Used to define variables to be used within the model.

**RESOURCE** Used to define a quantity of units of a resource that can be seized and released by entities.

**QUEUE** Used to define a waiting line for entities whose flow is currently stopped within the model.

**ENTITY** Used to define different entity types for use within the model.

Arena™ has many other facets that have yet to be touched upon. Not only does Arena™ allow the user to build and analyze simulation models, but it also makes experimentation easier with the Process Analyzer, which will run multiple simulation runs in one batch run and allow the comparison of these scenarios. In addition, Arena™ has the Input Analyzer to help to build models for the probability distributions used in a model. Finally, through its integration with OptQuest, Arena™ can assist in finding optimal design configurations via simulation.

The next chapter will dive deeper into how to use Arena™ to model process-oriented simulation situations.

## EXERCISES

- 4.1 *True or False:* The simulation clock for a discrete-event dynamic stochastic model jumps in equal increments of time in the defined time units. For example, 1 second, 2 seconds, 3 seconds, etc.
- 4.2 To debug an Arena™ model, the user can use the (a) \_\_\_\_\_, which can be found off of the (b) \_\_\_\_\_ menu item within the Arena™ environment.
- 4.3 Provide the missing concept or definition concerning an activity diagram.

Concept	Definition
(a)	Represented as a circle with a queue name labeled inside
(b)	Shown as a rectangle with an appropriate label inside
Resource	(c)
Lines/arcs	(d)
(e)	Indicates the creation or destruction of an entity

- 4.4 The service times for an automated storage and retrieval system has a shifted exponential distribution. It is known that it takes a minimum of 15 s for any retrieval. The parameter of the exponential distribution is  $\lambda = 45$ . Set up an Arena<sup>TM</sup> model that will generate 20 observations of the service times. Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.
- 4.5 The time to failure for a computer printer fan has a Weibull distribution with shape parameter  $\alpha = 2$  and scale parameter  $\beta = 3$ . Set up an Arena<sup>TM</sup> model that will generate 50 observations of the failure times. Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.
- 4.6 The time to failure for a computer printer fan has a Weibull distribution with shape parameter  $\alpha = 2$  and scale parameter  $\beta = 3$ . Testing has indicated that the distribution is limited to the range from 1.5 to 4.5. Set up an Arena<sup>TM</sup> model to generate 100 observations from this truncated distribution. Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.
- 4.7 The interest rate for a capital project is unknown. An accountant has estimated that the minimum interest rate will be between 2% and 5% within the next year. The accountant believes that any interest rate in this range is equally likely. You are tasked with generating interest rates for a cash flow analysis of the project. Set up an Arena<sup>TM</sup> model to generate 100 observations of the interest rate values for the capital project analysis. Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.
- 4.8 Develop an Arena<sup>TM</sup> model to generate 30 observations from the following probability density function:

$$f(x) = \begin{cases} \frac{3x^2}{2} & -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.

- 4.9 Suppose that the service time for a patient consists of two distributions. There is a 25% chance that the service time is uniformly distributed with a minimum of 20 min and a maximum of 25 min, and a 75% chance that the time is distributed according to a Weibull distribution with shape of 2 and a scale of 4.5.
- Set up an Arena<sup>TM</sup> model to generate 100 observations of the service time.
  - Compute the theoretical expected value of the distribution.
  - Estimate the expected value of the distribution and compute a 95% confidence interval on the expected value. Did your confidence interval contain the theoretical expected value of the distribution?
- 4.10 Suppose that  $X$  is a random variable with a  $N(\mu = 2, \sigma = 1.5)$  normal distribution. Generate 100 observations of  $X$  using an Arena<sup>TM</sup> model.
- Estimate the mean from your observations. Report a 95% confidence interval for your point estimate.
  - Estimate the variance from your observations. Report a 95% confidence interval for your point estimate.
  - Estimate the  $P\{X \geq 3\}$  from your observations. Report a 95% confidence interval for your point estimate.

- 4.11 Samples of 20 parts from a metal grinding process are selected every hour. Typically 2% of the parts needs rework. Let  $X$  denotes the number of parts in the sample of 20 that require rework. A process problem is suspected if  $X$  exceeds its mean by more than 3 standard deviations. Using an Arena™ model, simulate 30 hours of the process, that is, 30 samples of size 20, and estimate the chance that  $X$  exceeds its expected value by more than 3 standard deviations.
- 4.12 Consider the following discrete distribution of the random variable  $X$  whose probability mass function is  $p(x)$ .

$x$	0	1	2	3	4
$p(x)$	0.3	0.2	0.2	0.1	0.2

- Set up an Arena™ model to generate 30 observations of the random variable  $X$ . Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.

- 4.13 The demand for parts at a repair bench per day can be described by the following discrete probability mass function:

Demand	0	1	2
Probability	0.3	0.2	0.5

- Set up an Arena™ model to generate 30 observations of the demand. Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.

- 4.14 Using Arena™ and the Monte-Carlo method, estimate the following integral with 95% confidence to within  $\pm 0.01$ .

$$\int_1^4 \left( \sqrt{x} + \frac{1}{2\sqrt{x}} \right) dx$$

- 4.15 Using Arena™ and the Monte-Carlo method, estimate the following integral with 99% confidence to within  $\pm 0.01$ .

$$\int_0^\pi (\sin(x) - 8x^2) dx$$

- 4.16 Using Arena™ and the Monte-Carlo method, estimate the following integral with 99% confidence to within  $\pm 0.01$ .

$$\theta = \int_0^1 \int_0^1 (4x^2y + y^2) dx dy$$

- 4.17 A firm is trying to decide whether or not it should purchase a new scale to check a package filling line in the plant. The scale would allow for better control over the filling operation and result in less overfilling. It is known for certain that the scale costs \$800 initially. The annual cost has been estimated to be normally distributed

with a mean of \$100 and a standard deviation of \$10. The extra savings associated with better control of the filling process has been estimated to be normally distributed with a mean of \$300 and a standard deviation of \$50. The salvage value has been estimated to be uniformly distributed between \$90 and \$100. The useful life of the scale varies according to the amount of usage of the scale. The manufacturing has estimated that the useful life can vary between 4 and 7 years with the chances given in the following table.

years	4	5	6	7
f(years)	0.3	0.4	0.1	0.2
F(years)	0.3	0.7	0.8	1.0

The interest rate has been varying recently and the firm is unsure of the rate for performing the analysis. To be safe, they have decided that the interest rate should be modeled as a beta random variable over the range from 6% to 9% with alpha = 5.0 and beta = 1.5. Given all the uncertain elements in the situation, they have decided to perform a simulation analysis in order to assess the expected present value of the decision and the chance that the decision has a negative return.

We desire to be 95% confident that our estimate of the true expected present value is within  $\pm 10$  dollars. Develop an Arena™ model for this situation.

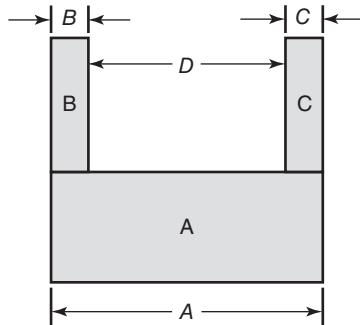
- 4.18 A firm is trying to decide whether or not to invest in two proposals A and B that have the net cash flows shown in the following table, where  $N(\mu, \sigma)$  represents that the cash flow value comes from a normal distribution with the provided mean and standard deviation.

End of Year	0	1	2	3	4
A	$N(-250, 10)$	$N(75, 10)$	$N(75, 10)$	$N(175, 20)$	$N(150, 40)$
B	$N(-250, 5)$	$N(150, 10)$	$N(150, 10)$	$N(75, 20)$	$N(75, 30)$

The interest rate has been varying recently and the firm is unsure of the rate for performing the analysis. To be safe, they have decided that the interest rate should be modeled as a beta random variable over the range from 2% to 7% with  $\alpha_1 = 4.0$  and  $\alpha_2 = 1.2$ . Given all the uncertain elements in the situation, they have decided to perform a simulation analysis in order to assess the situation. Use Arena™ to answer the following questions:

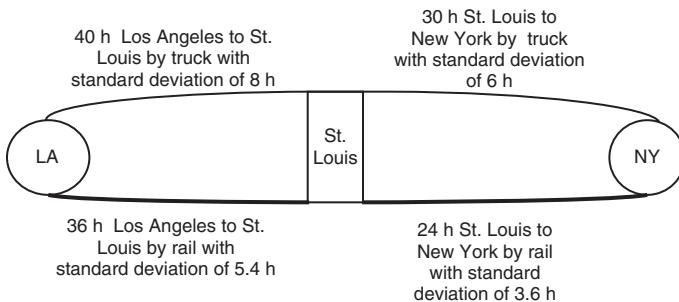
- (a) Compare the expected present worth of the two alternatives. Estimate the probability that alternative A has a higher present value than alternative B.
  - (b) Determine the number of samples needed to be 95% confidence that you have estimated the  $P\{PV(A) > PV(B)\}$  to within  $\pm 0.10$ , where  $PV(A)$  and  $PV(B)$  represent the present value of alternatives A and B, respectively.
- 4.19 A U-shaped component is to be formed from the three parts A, B, and C. The picture is shown in the figure below. The length of A is lognormally distributed with a mean

of 20 mm and a standard deviation of 0.2 mm. The thickness of parts B and C is uniformly distributed with a minimum of 4.98 mm and a maximum of 5.02 mm. Assume all dimensions are independent.



Develop an Arena™ model to estimate the probability that the gap  $D$  is less than 10.1 mm with 95% confidence to within  $\pm 0.01$  mm.

- 4.20 Shipments can be transported by rail or trucks between New York and Los Angeles. Both modes of transport go through the city of St. Louis. The mean travel time and standard deviations between the major cities for each mode of transportation are shown in the following figure.



Assume that the travel times (in either direction) are lognormally distributed as shown in the figure. For example, the time from New York to St. Louis (or St. Louis to New York) by truck is 30 h with a standard deviation of 6 h. In addition, assume that the transfer time in hours in St. Louis is triangularly distributed with parameters (8, 10, 12) for trucks (truck to truck). The transfer time in hours involving rail is triangularly distributed with parameters (13, 15, 17) for rail (rail to rail, rail to truck, truck to rail). We are interested in determining the shortest total shipment time combination from New York to Los Angeles. Develop an Arena™ simulation for this problem.

- How many shipment combinations are there?
- Write an Arena™ expression for the total shipment time of the truck only combination.
- We are interested in estimating the average shipment time for each shipment combination and the probability that the shipment combination will be able to deliver the shipment within 85 h.

- (d) Estimate the probability that a shipping combination will be the shortest.
- (e) Determine the sample size necessary to estimate the mean shipment time for the truck only combination to within 0.5 h with 95% confidence.
- 4.21 A firm produces YBox gaming stations for the consumer market. Their profit function is

$$\text{Profit} = (\text{Unit price} - \text{Unit cost}) \times (\text{Quantity sold}) - \text{Fixed costs}$$

Suppose that the unit price is \$200 per gaming station and that the other variables have the following probability distributions:

Unit cost	80	90	100	110
Probability	0.20	0.40	0.30	0.10
Quantity sold	1000	2000	3000	
Probability	0.10	0.60	0.30	
Fixed cost	50,000	65,000	80,000	
Probability	0.40	0.30	0.30	

Use an Arena™ model to generate 1000 observations of the profit.

- (a) Estimate the mean profit from your sample and compute a 95% confidence interval for the mean profit.
- (b) Estimate the probability that the profit will be positive.
- 4.22 T. Wilson operates a sports magazine stand before each game. He can buy each magazine for 33 cents and can sell each magazine for 50 cents. Magazines not sold at the end of the game are sold for scrap for 5 cents each. Magazines can only be purchased in bundles of 10. Thus, he can buy 10, 20, and so on magazines prior to the game to stock his stand. The lost revenue for not meeting demand is 17 cents for each magazine demanded that could not be provided. Mr. Wilson's profit is as follows:

$$\begin{aligned} \text{Profit} &= (\text{Revenue from sales}) - (\text{Cost of magazines}) \\ &\quad - (\text{Lost profit from excess demand}) \\ &\quad + (\text{Salvage value from sale of scrap magazines}) \end{aligned}$$

Not all game days are the same in terms of potential demand. The type of day depends on a number of factors including the current standings, the opponent, and whether or not there are other special events planned for the game day weekend. There are three types of game days demand: high, medium, and low. The type of day has a probability distribution associated with it.

Type of day	High	Medium	Low
Probability	0.35	0.45	0.20

The amount of demand for magazines then depends on the type of day according to the following distributions:

Demand	High		Medium		Low	
	PMF	CDF	PMF	CDF	PMF	CDF
40	0.03	0.03	0.1	0.1	0.44	0.44
50	0.05	0.08	0.18	0.28	0.22	0.66
60	0.15	0.23	0.4	0.68	0.16	0.82
70	0.2	0.43	0.2	0.88	0.12	0.94
80	0.35	0.78	0.08	0.96	0.06	1.0
90	0.15	0.93	0.04	1.0		
100	0.07	1.0				

Let  $Q$  be the number of units of magazines purchased (quantity on hand) to set up the stand. Let  $D$  represent the demand for the game day. If  $D > Q$ , Mr. Wilson sells only  $Q$  and will have lost sales of  $D-Q$ . If  $D < Q$ , Mr. Wilson sells only  $D$  and will have scrap of  $Q-D$ . Assume that he has determined that  $Q = 50$ .

Make sure that you can estimate the average profit and the probability that the profit is greater than zero for Mr. Wilson. Develop an Arena™ model to estimate the average profit with 95% confidence to within  $\pm \$0.5$ .

- 4.23 The time for an automated storage and retrieval system in a warehouse to locate a part consists of three movements. Let  $X$  be the time to travel to the correct aisle. Let  $Y$  be the time to travel to the correct location along the aisle. And let  $Z$  be the time to travel up to the correct location on the shelves. Assume that the distributions of  $X$ ,  $Y$ , and  $Z$  are as follows:

- $X \sim \text{lognormal}$  with mean 20 and standard deviation 10 seconds
- $Y \sim \text{uniform}$  with minimum 10 and maximum 15 seconds
- $Z \sim \text{uniform}$  with minimum of 5 and a maximum of 10 seconds

Develop an Arena™ model that can estimate the average total time that it takes to locate a part and can estimate the probability that the time to locate a part exceeds 60 seconds. Base your analysis on 1000 observations.

- 4.24 Lead-time demand may occur in an inventory system when the lead-time is other than instantaneous. The lead-time is the time from the placement of an order until the order is received. The lead-time is a random variable. During the lead-time, demand also occurs at random. Lead-time demand is thus a random variable defined as the sum of the demands during the lead-time, or  $LDT = \sum_{i=1}^T D_i$ , where  $i$  is the time period of the lead-time and  $T$  is the lead-time. The distribution of lead-time demand is determined by simulating many cycles of lead-time and the demands that occur during the lead-time to get many realizations of the random variable LDT. Notice that LDT is the *convolution* of a random number of random demands. Suppose that the daily demand for an item is given by the following probability mass function:

Daily demand (items)	4	5	6	7	8
Probability	0.10	0.30	0.35	0.10	0.15

The lead-time is the number of days from placing an order until the firm receives the order from the supplier.

- (a) Assume that the lead-time is a constant 10 days. Develop an Arena™ model to simulate 1000 instances of LDT. Report the summary statistics for the 1000 observations. Estimate the chance that LDT is greater than or equal to 10. Report a 95% confidence interval on your estimate.
  - (b) Assume that the lead-time has a shifted geometric distribution with probability parameter equal to 0.2. Use an Arena™ model to simulate 1000 instances of LDT. Report the summary statistics for the 1000 observations. Estimate the chance that LDT is greater than or equal to 10. Report a 95% confidence interval on your estimate.
- 4.25 If  $Z \sim N(0, 1)$ , and  $Y = \sum_{i=1}^k Z_i^2$  then  $Y \sim \chi_k^2$ , where  $\chi_k^2$  is a chi-squared random variable with  $k$  degrees of freedom. Set up an Arena™ model to generate 50  $\chi_5^2$  random variates. Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.
- 4.26 Set up an Arena™ model that will generate 30 observations from the following probability density function using the acceptance–rejection algorithm for generating random variates.

$$f(x) = \begin{cases} \frac{3x^2}{2} & -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

- 4.27 This procedure is due to Box and Muller [1958]. Let  $U_1$  and  $U_2$  be two independent uniform  $(0,1)$  random variables and define

$$\begin{aligned} X_1 &= \cos(2\pi U_2)\sqrt{-2 \ln(U_1)} \\ X_2 &= \sin(2\pi U_2)\sqrt{-2 \ln(U_1)} \end{aligned}$$

It can be shown that  $X_1$  and  $X_2$  will be independent standard normal random variables, that is,  $N(0, 1)$ . Use Arena™ to implement the Box and Muller algorithm for generating normal random variables. Generate 1000  $N(\mu = 2, \sigma = 0.75)$  random variables via this method. Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.

- 4.28 Consider the banking situation of Section 4.4. A simulation analyst observed the operation of the bank and recorded the information given in the following table. The information was recorded right after the bank opened during a period of time for which there was only one teller working. From this information, you would like to recreate the operation of the system.

Customer Number	Time of Arrival	Service Time
1	3	4
2	11	4
3	13	4
4	14	3
5	17	2
6	19	4
7	21	3
8	27	2
9	32	2
10	35	4
11	38	3
12	45	2
13	50	3
14	53	4
15	55	4

The following table presents the operation of the system for the first four customers. The row for time 0 indicates that there were no customers present and that the teller was idle. The second row marks the time of the first event, which happens to be the arrival of the first customer at time 3.0

(1) Event Time	(2) Cust. #	(3) Event Time	(4) Arrival Time	(5) Start Service	(6) Depart Time	(7) Time in System	(8) # in Queue	(9) Teller Status	(10) Idle Time
0	---	---	---	---	---	---	0	I	---
3	1	A	3	3	7		0	B	3
7	1	D				4	0	I	
11	2	A	11	11	15		0	B	4
13	3	A	13	15	19		1	B	
14	4	A	14	19	22		2	B	

The columns of the table denote specifics for what happened at the given time. For example, from the table we know that customer 1 arrived at time 3.0 and that the customer started service at time 3 (because the teller was idle). We also know that the first customer had a service time of 4 min. Thus, we can compute column (6) which indicates when the customer will depart. The time in the system, column (7), should be computed when the customer actually leaves the system. When the customer arrived at time 3.0, there were 0 customers in the queue and the teller became busy as shown in columns (8) and (9). Since the teller became busy, we can compute how long the

teller had been idle as shown in column (10). There were no arrivals prior to time 7, which is when the first customer should depart. Thus, the third row indicates the next event as customer 1 departing. At this event, the system time, column (7) can be computed (time of departure time of arrival =  $7 - 3 = 4$ ). In addition, since there are no customers in the queue, the teller's status becomes idle in column (9). Verify that you understand how rows for the event times of 11, 13, and 15 are tabulated. Complete the table and compute the average of the system times for the customers. What percentage of the total time was the teller idle?

- 4.29 Consider again the arrival and service data from Exercise 4.28. Continue the simulation of all the arrivals until all arrivals are processed.
- Compute the average time spent in the system for the customers.
  - Compute the percentage of time that the teller was idle.
  - Compute the percentage of time that there were 0, 1, 2, and 3 customers in the queue.
- 4.30 Consider the following interarrival and service times for the first 25 customers to a single server queuing system.

Customer Number	Interarrival Time	Service Time	Time of Arrival
1	22	74	22
2	89	105	111
3	21	34	132
4	26	38	158
5	80	23	
6	81	26	
7	78	90	
8	20	26	
9	32	37	
10	13	88	
11	28	38	
12	18	73	
13	29	93	
14	19	25	
15	20	93	
16	23	5	
17	78	37	
18	20	51	
19	109	28	
20	78	85	

The following table presents the operation of the system for the first few customers. The row for time 0 indicates that there were no customers present and that the server was idle. The second row marks the time of the first event, which happens to be the arrival of the first customer at time 22. The columns of the table denote specifics for what happened at the given time. For example, we know that customer 1 arrived at time 22 and that the customer started service at time 22 (because the server was

idle). We also know that the first customer had a service time of 74 seconds. Thus, we can compute column (6) which indicates when the customer will depart. The time in the system, column (7), should be computed when the customer actually leaves the system. When the customer arrived at time 22, there were 0 customers in the queue and the teller became busy as shown in columns (8) and (9). Since the teller became busy, we can compute how long the teller had been idle as shown in column (10). The departure time computed in column (6) indicates that a departure event will occur at time 96. Since the next customer does not arrive until time  $22 + 89 = 111$ , the departure event is the next event to occur.

The next arrival is to occur 111, with service complete at 216. The next arrival after that is at 132. Thus, the customer will complete service before the next arrival. The customer that arrives at 132 has a service time of 34. The departure will be at time 250. The next customer to arrive will arrive at 158. What happens to this customer? Complete the rest of the table for all the customers.

(1) Event Time	(2) Cust. #	(3) Event Time	(4) Arrival Time	(5) Start Service	(6) Depart Time	(7) Time in System	(8) # in Queue	(9) Teller Status	(10) Idle Time
0	—	—	—	—	—	—	0	0	—
22	1	A	22	22	$22 + 74 = 96$		0	1	22
96	1	D				$96 - 22 = 74$	0	0	
111	2	A	111	111	$111 + 105 = 21.6$		0	1	$111 - 96 = 15$
132	3	A	132	216	$216 + 34 = 25$		1	1	0
158	4	A	158	250	$250 + 38 = 288$		2	1	0
216	2	D				$216 - 111 = 105$	1	1	
238	5	A	238	288	311		2	1	0

- (a) We are given the interarrival times. Determine the time of arrival of each customer.
  - (b) Complete the event and state variable change table associated with this situation.
  - (c) Draw a sample path graph for the variable  $N(t)$  which represents the number of customers in the system at any time  $t$ . Compute the average number of customers in the system over the time period from 0 to 700.
  - (d) Draw a sample path graph for the variable  $NQ(t)$  which represents the number of customers waiting for the server at any time  $t$ . Compute the average number of customers in the queue over the time period from 0 to 700.
  - (e) Draw a sample path graph for the variable  $B(t)$  which represents the number of servers busy at any time  $t$ . Compute the average number of busy servers in the system over the time period from 0 to 700.
  - (f) Compute the average time spent in the system for the customers.
- 4.31 Parts arrive at a station with a single machine according to a Poisson process with the rate of 1.5 per minute. The time it takes to process the part has an exponential distribution with a mean of 30 seconds. There is no upper limit on the number of parts that wait for process. Set up an Arena™ model to estimate the expected number of parts waiting in the queue and the utilization of the machine. Run your model for

10,000 seconds for 30 replications and report the results. Use Equation (4.2) to verify that your simulation is working as intended.

- 4.32 A large car dealer has a policy of providing cars for its customers who have car problems. When a customer brings the car in for repair, that customer has use of a dealer's car. The dealer estimates that the dealer cost for providing the service is \$10 per day for as long as the customer's car is in the shop. Thus, if the customer's car was in the shop for 1.5 days, the dealer's cost would be \$15. Arrivals to the shop of customers with car problems form a Poisson process with a mean rate of one every other day. There is one mechanic dedicated to the customer's car. The time that the mechanic spends on a car can be described by an exponential distribution with a mean of 1.6 days. Set up an Arena™ model to estimate the expected time within the shop for the cars and the utilization of the mechanic. Run your model for 10,000 days for 30 replications and report the results. Estimate the total cost per day to the dealer for this policy. Use Equation (4.2) to verify that your simulation is working as intended.
- 4.33 YBox video game players arrive according to a Poisson process with rate 10 per hour to a two-person station for inspection. The inspection time per YBox set is EXPO(10) minutes. On an average 82% of the sets pass inspection. The remaining 18% is routed to an adjustment station with a single operator. Adjustment time per YBox is UNIF(7,14) minutes. After adjustments are made, the units depart the system. The company is interested in the total time spent in the system. Run your model for 10,000 minutes for 30 replications and report the results.
- 4.34 The lack of memory property of the exponential distribution states that given  $\Delta t$  is the time period that elapsed since the occurrence of the last event, the time  $t$  remaining until the occurrence of the next event is independent of  $\Delta t$ . This implies that  $P\{X > \Delta t + t | X > t\} = P\{X > t\}$ . Describe a simulation experiment that would allow you to test the lack of memory property empirically. Implement your simulation experiment in Arena™ and test the lack of memory property empirically. Explain in your own words what lack of memory means.



---

# 5

---

## BASIC PROCESS MODELING

### LEARNING OBJECTIVES

- To be able to define and explain the key elements of process-oriented simulations.
- To be able to identify entities and their attributes and use them within Arena<sup>TM</sup>.
- To be able to identify, define, and use variables within Arena<sup>TM</sup>.
- To be able to use Arena<sup>TM</sup> to perform basic input/output operations.
- To be able to program flow of control within an Arena<sup>TM</sup> model.
- To be able to understand the programming aspects of Arena<sup>TM</sup> including debugging.
- To be able to understand and model shared resources.
- To be able to understand and model the batching and separation of entities.

### 5.1 ELEMENTS OF PROCESS-ORIENTED SIMULATION

Chapter 1 described a system as a set of interrelated components that work together to achieve common objectives. In this chapter, a method for modeling the operation of a system by describing its processes is presented. In the simplest sense, a process can be thought of as a sequence of activities, where an activity is an element of the system that takes an interval of time to complete. In the pharmacy example, the service of the customer by the pharmacist was an activity. The representation of the dynamic behavior of the system by describing the process flows of the entities moving through the system is called process-oriented modeling. When developing a simulation model using the process view, there are a number of terms

and concepts that are often used. Before learning some of these concepts in more detail, it is important that you begin with an understanding of some of the vocabulary used within simulation. The following terms will be used throughout the text:

**System** A set of interrelated components that act together over time to achieve common objectives.

**Parameters** Quantities that are properties of the system that do not change. These are typically quantities (variables) that are part of the environment that the modeler feels cannot be controlled or changed. Parameters are typically model inputs in the form of variables.

**Variables** Quantities that are properties of the system (as a whole) that change or are determined by the relationships between the components of the system as it evolves through time.

**System State** A “snap shot” of the system at a particular point in time characterized by the values of the variables that are necessary for determining the future evolution of the system from the present time. The minimum set of variables that are necessary to describe the future evolution of the system is called the system’s state variables.

**Entity** An object of interest in the system whose movement or operation within the system may cause the occurrence of events.

**Attribute** A property or variable that is associated with an entity.

**Event** An instantaneous occurrence or action that changes the state of the system at a particular point in time.

**Activity** An interval of time bounded by two events (start event and end event).

**Resource** A limited quantity of items that are used (e.g., seized and released) by entities as they proceed through the system. A resource has a capacity that governs the total quantity of items that may be available. All the items in the resource are homogeneous, meaning that they are indistinguishable. If an entity attempts to seize a resource that does not have any units available, it must wait in a queue.

**Queue** A location that holds entities when their movement is constrained within the system.

**Future Event List** A list that contains the time-ordered sequence of events for the simulation.

When developing models, it will be useful to identify the elements of the system that fit some of these definitions. An excellent place to develop your understanding of these concepts is with entities because process-oriented modeling is predicated on describing the life of an entity as it moves through the system.

## 5.2 ENTITIES, ATTRIBUTES, AND VARIABLES

When modeling a system, there are often many types of entities. For example, consider a retail store. Besides customers, the products might also be considered as entities. The products are received by the store and “wait” on the shelves until customers select them for purchase. Entities may come in groups and then are processed individually or they might start out as individual units that are formed into groups. For example, a truck arriving to the

store may be an entity that consists of many pallets that contain products. The customers select the products from the shelves, and during the check out process, the products are placed in bags. The customers then carry their bags to their cars. Entities are uniquely identifiable within the system. If there are two customers in the store, they can be distinguished by the *values* of their attributes. For example, considering a product as an entity, it may have attributes *serial number*, *weight*, *category*, and *price*. The set of attributes for a type of entity is called its *attribute set*. While all products might have these attributes, they do not necessarily have the same values for each attribute. For example, consider the following two products:

- (serial number = 12345, weight = 8 ounces, category = green beans, price = \$0.87)
- (serial number = 98765, weight = 8 ounces, category = corn, price = \$1.12)

The products carry or retain these attributes and their values as they move through the system. In other words, attributes are attached to or associated with entities. The values of the attributes might change during the operation of the system. For example, a mark down on the price of green beans might occur after some period of time. Attributes can be thought of as variables that are attached to entities.

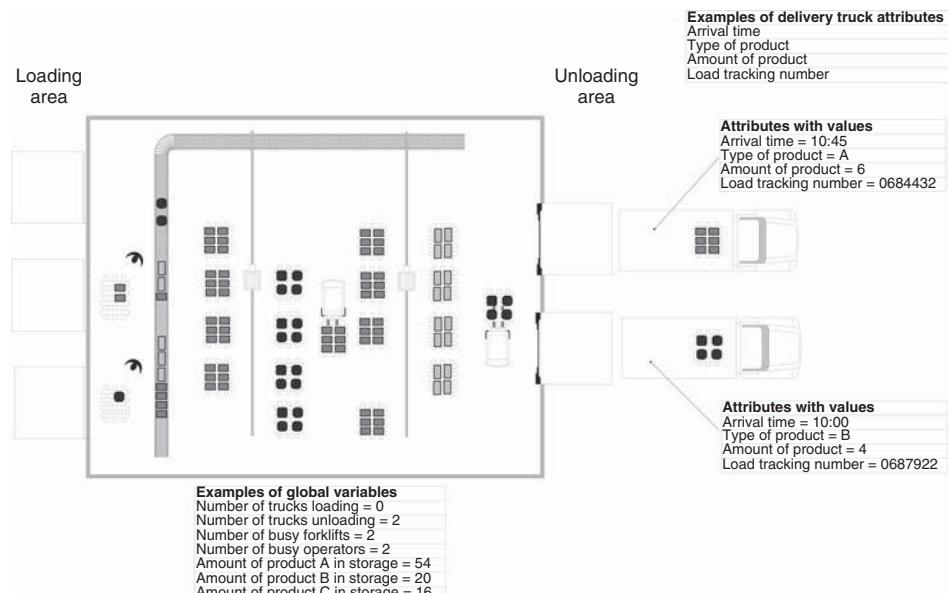
Not all information in a system is local to the entities. For example, the number of customers in the store, the number of carts, and the number of check out lanes are all characteristics of the system. These types of data are called *system attributes*. In simulation models, this information can be modeled with global *variables* or other data modules (e.g., resources) to which all entities can have access. By making these quantities visible at the system level, the information can be shared between the entities within the model and between the components of the system.

Figure 5.1 illustrates the difference between global (system) variables and entities with their attributes in the context of a warehouse. In the figure, the trucks are entities with attributes: arrival time, type of product, amount of product, and load tracking number. Notice that both of the trucks have these attributes, but each truck has different *values* for their attributes. The figure also illustrates examples of global variables, such as number of trucks loading, number of trucks unloading, and number of busy forklifts. This type of information belongs to the whole system.

Once a basic understanding of the system is accomplished through understanding system variables, the entities, and their attributes, you must start to understand the processes within the system. Developing the process description for the various types of entities within a system and connecting the flow of entities to the changes in state variables of the system are the essence of process-oriented modeling. In order for entities to flow through the model and experience processes, you must be able to create (and dispose of) the entities. The following section describes how Arena<sup>TM</sup> allows the modeler to create and dispose of entities.

### **5.3 CREATING AND DISPOSING OF ENTITIES**

The basic mechanism by which entities are introduced into an Arena<sup>TM</sup> model is the CREATE module. An entity is an object that flows within the model. As an entity flows through the model it causes the *execution of each module through which it flows*. Because of this,



**Figure 5.1** Global variables and attributes within a system.

nothing will happen in an Arena™ model unless entities are created.<sup>1</sup> Entities can be used to represent instances of actual physical objects that move through the system. For example, in a model of a manufacturing system, entities that represent the parts that are produced by the system will need to be created. Sometimes entities do not have a physical representation in the system and are simply used to represent some sort of logical use. For example, an entity whose sole purpose is to generate random numbers or to read data from an input file may be created within a model. You will learn various ways to model with entities as you proceed through the text.

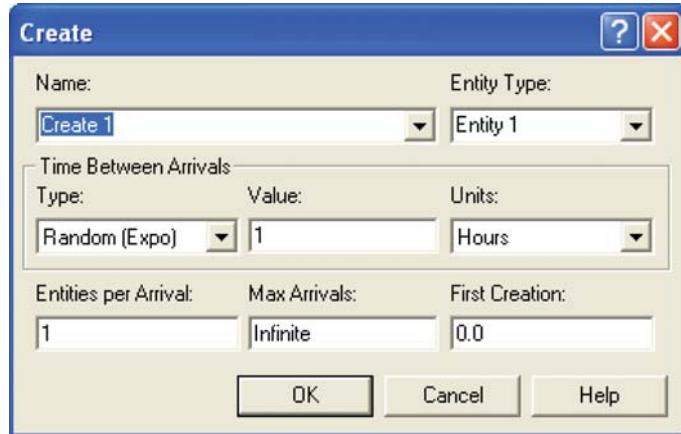
Figure 5.2 illustrates the CREATE module dialog box that opens when double-clicking on the flowchart symbol within the model window. Understanding what the dialog box entries mean is essential to writing Arena™ models. Each dialog box has a Help button associated with it. Clicking on this button will reveal help files that explain the basic functionality of the module. In addition, the text field prompts are explained within the help system.

According to the Arena™ help files, the basic dialog entries are as follows:

**Type** Type of arrival stream to be generated. Types include *Random* (uses an exponential distribution, user specifies the mean of the distribution), *Schedule* (specifies a nonhomogeneous Poisson process with rates determined from the specified Schedule module), *Constant* (user specifies a constant value, e.g., 100), and *Expression* (pull down list of various distributions or general expressions written by the user).

**Value** Determines the mean of the exponential distribution (if *Random* is used) or the constant value (if *Constant* is used) for the time between arrivals. Applies only when

<sup>1</sup>Actually, this statement is not quite true and not quite false. You do not necessarily have to have a CREATE module in the model to get things to happen, but this requires the use of advanced techniques that are beyond the scope of this book.



**Figure 5.2** CREATE module flowchart symbol and dialog box.

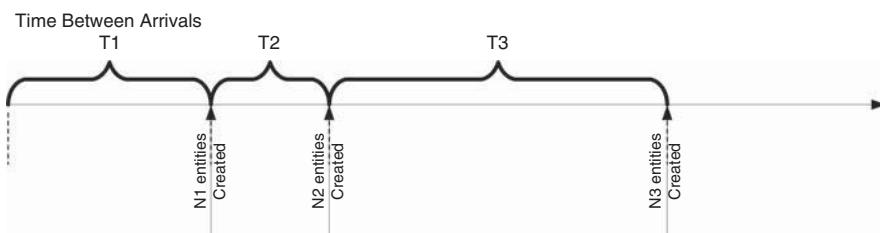
Type is *Random* or *Constant*. Can be a general expression when the type is specified as Expression.

**Entities per Arrival** Number of entities that will enter the system at a given time with each arrival. This allows for the modeling of batch arrivals.

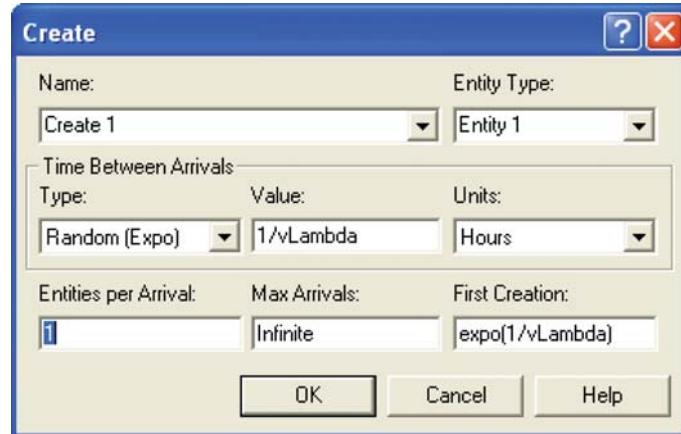
**Max Arrivals** Maximum number of entities that the module will generate. When this value is reached, the creation of new entities by the module ceases.

**First Creation** Starting time for the first entity to arrive into the system. Does not apply when Type is *Schedule*.

The CREATE module defines a repeating pattern for an arrival process. The time between arrivals is governed by the specification of the type of arrival stream. The time between arrivals specifies an ordered sequence of events in time at which entities are created and introduced to the model. At each event, a number of entities can be created. The number of entities created is governed by the “Entities per Arrival” field, which may be stochastic. The first arrival is governed by the specification of the “First Creation” time, which also may be stochastic. The maximum number of arrival events is governed by the “Max Arrivals” entry. Figure 5.3 illustrates an arrival process where the time of the first arrival is given by  $T_1$ , the time of the second arrival is given by  $T_1 + T_2$ , and the time of the third arrival is given by  $T_1 + T_2 + T_3$ . In the figure, the number of arriving entities at each arriving event is given by  $N_1$ ,  $N_2$ , and  $N_3$  respectively.



**Figure 5.3** Example arrival process.



**Figure 5.4** CREATE module for Poisson process.

For example, to specify a Poisson arrival process with mean rate  $\lambda$ , a CREATE module as shown in Figure 5.4 can be used. Why does this specify a Poisson arrival process? It is because the time between arrivals for a Poisson process with rate  $\lambda$  is exponentially distributed with the mean of the exponential distribution being  $1/\lambda$ .

To specify a compound arrival process, use the “Entities per Arrival” field. For example, suppose you have a compound Poisson process<sup>2</sup> where the distribution for the number created at each arrival is governed by a discrete distribution.

$$P(X = x) = \begin{cases} 0.2 & x = 1 \\ 0.3 & x = 2 \\ 0.5 & x = 3 \end{cases}$$

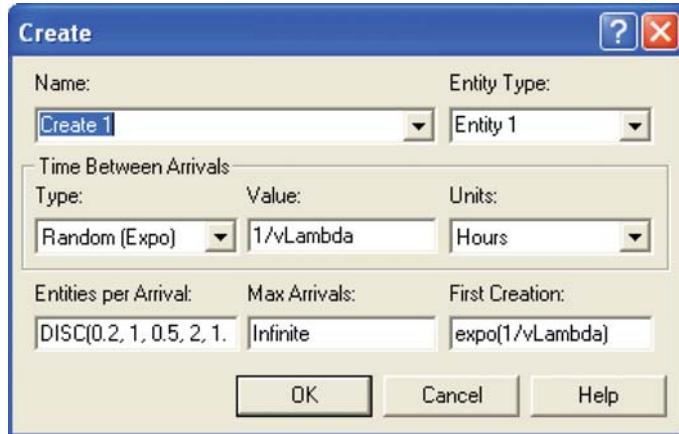
Figure 5.5 shows the CREATE module for this compound Poisson process with the Entities per Arrival field using the DISC(0.2, 1, 0.5, 2, 1.0, 3) discrete empirical distribution function.

When developing models, it is often useful to create one entity and use that entity to trigger other actions in the model. For example, to read in information from a file, a logical entity can be created at time zero and used to execute a READ module. To create one and only one entity at time 0, specify the *Max Arrivals* field as one, the *First Creation* field as 0.0, the *Entities per Arrival* as one, and the type field as Constant. The value specified for the constant will be immaterial since only one entity will be created.

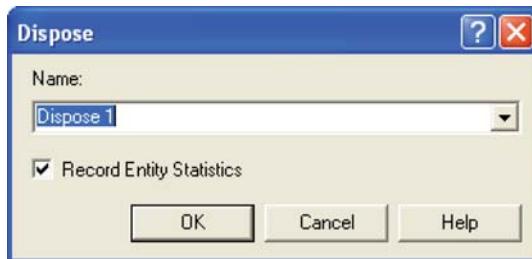
Each entity that is created allocates memory for the entity. Once the entity has traveled through its process within the model, the entity should be disposed. The DISPOSE module acts as a “sink” to dispose of entities when they are no longer needed within the model.

Figure 5.6 illustrates the dialog box for the DISPOSE module. The DISPOSE module indicates the number of entities that are disposed by the module within the animation. In addition, the default action is to record the entity statistics prior to disposing the entity. If the Entities Statistics Collection field is checked within the Project Parameters tab of the Run/Setup dialog, the entity statistics will include VA Time (value added time), NVA Time

<sup>2</sup>See Ross (1997) for more information on the theory of compound Poisson processes.



**Figure 5.5** CREATE module for compound Poisson process.



**Figure 5.6** DISPOSE module flowchart symbol and dialog box.

(nonvalue added time), Wait Time, Transfer Time, Other Time, and entity Total Time. If the Costing Statistics Collection field is also checked within the Project Parameters tab of the Run/Setup dialog, additional entity statistics include VA Cost (value added cost), NVA Cost (nonvalue added cost), Wait Cost, Transfer Cost, Other Cost, and Total Cost. Additionally, the number of entities leaving the system (for a given entity type) and currently in the system (WIP, work in process) will be tabulated. Chapter 10 will describe the use of some of these attributes within the context of Arena’s cost modeling system.

You now know how to introduce entities into a model and how to dispose of entities after you are done with them. To make use of the entities within the model, you must first understand how to define and use variables and entity attributes within a model.

#### 5.4 DEFINING VARIABLES AND ATTRIBUTES

In a programming language like C, variables must first be defined before using them in a program. For example, in C, a variable named *x* can be defined as type float and then used in assignment statements such as *float x; x = x + 2;*

Variables in standard programming languages have a specific scope associated with their execution. For example, variables defined within a function are limited to use within that function. Global variables that can be used throughout the entire program may also be defined.

In Arena™, *all variables are global*. You can think of variables as belonging to the system being modeled. Everything in the system can have access to the variables of the system. Variables provide a named location in computer memory that persists until changed during execution. The naming conventions for declaring variables in Arena™ are quite liberal. This can be both a curse and a blessing. Because of this, you should adopt a standard naming convention when defining variables. The models in this text try to start the name of all variables with the letter “v.” For example, in the previously discussed CREATE module examples, the variable *vLambda* was used to represent the arrival rate of the Poisson process. This makes it easy to distinguish variables from other Arena™ constructs when reviewing, debugging, and documenting the model.

In addition, the naming rules allow the use of spaces within a variable name, for example, *This is a legal name* is a legally named variable within Arena™. Suppose you needed to count the number of parts and decided to name your variable *Part Count*. The use of spaces in this manner should be discouraged because woe unto you if you have this problem: *Part Count* and *Part Count*. Can you see the problem? The second variable *Part Count* has extra spaces between *Part* and *Count*. Try searching through every dialog box to find that in a model!

According to the naming convention recommended here, you would name this variable *vPartCount*, concatenating and capitalizing the individual components of the name. Of course you are free to name your variables whatever you desire as long as the names do not conflict with already existing variables or keywords within Arena™. To learn more about the specialized variables available within Arena™, you should refer to the portable document file (PDF) document *Arena™ Variables Guide*, which comes with the Arena™ installation. Reading this document should give you a better feeling for the functional capabilities of Arena™.

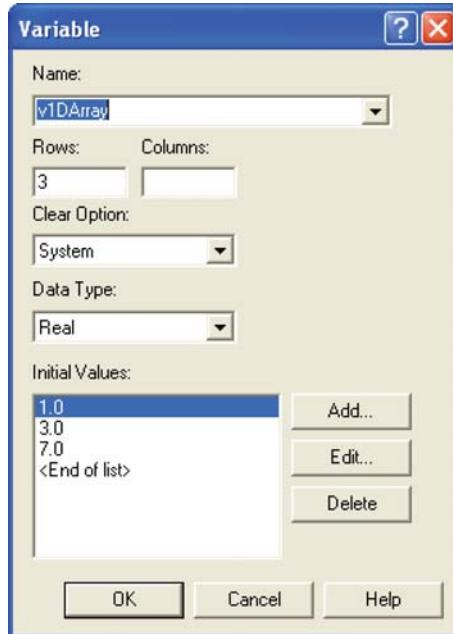
To declare variables, you use the VARIABLE module found in the Basic Process template (Figure 5.7). The VARIABLE module is a data module and is accessed either through the spreadsheet view or through a dialog box (Figure 5.8).

Within Arena™, variables can be scalars or can be defined as arrays. The arrays can be either one dimensional or two dimensional. For one-dimensional arrays, you specify either the number of rows or the number of columns. There is no difference between an array specified with, say three rows, or an array specified with three columns. The net effect is that there will be three variables defined which can be indexed by the numbers 1, 2, and 3. When defining a two-dimensional array of variables, you specify the number of rows and columns. Array indexes start at 1 and run through the size of the specified dimension. A runtime error will occur if the index used to access an array is not within the bounds of the array’s dimensions.

Variable - Basic Process							
	Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics
1	vScalar			Real	System	0 rows	<input type="checkbox"/>
2	v1DArray	3		Real	System	3 rows	<input type="checkbox"/>
3	v2DArray	2	5	Real	System	10 rows	<input type="checkbox"/>

Double-click here to add a new row.

Figure 5.7 Spreadsheet view of VARIABLE data module.



**Figure 5.8** Dialog box for defining variables.

Initial Values						
	1	2	3	4	5	
1	1.1	1.2	3.1	9.0	99.9	
2	10.5	4.3	0.0	2.9	27.4	

Variable - Basic Process							
	Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics
1	vScalar			Real	System	0 rows	<input type="checkbox"/> Click to enter values
2	v1DArray	3		Real	System	3 rows	<input type="checkbox"/>
3	v2DArray	2	5	Real	System	10 rows	<input type="checkbox"/>

Double-click here to add a new row.

**Figure 5.9** Spreadsheet view for 2D array.

When defining a variable or an array of variables, Arena™ allows you to specify the initial value(s). The default initial value for all variables is zero. By using the spreadsheet view within the Data window, see Figure 5.9, you can easily enter the initial values for variables and arrays. All variables are treated as real numbers within Arena™. If you need to represent an integer, for example, *vPartCount*, you simply represent it within Arena™ as a variable.

It is important to remember that all variables or arrays are global within Arena™. Variables are used to share information across all modules within the model. You should think of variables as characteristics or properties of the model as a whole. Variables belong to the entire system being modeled.

Arena™ has another data type that is capable of holding values called an attribute. An attribute is a named property or characteristic of an entity. For example, suppose the time to produce a part depends on the dimensions or area associated with the part. A CREATE module can be used to create entities that represent the parts. After the parts are created, the area associated with each part needs to be set. Each part created could have different values for its area attribute.

- Part 1: area = 5 square inches
- Part 2: area = 10 square inches

Part 1 and part 2 have the same named attribute, area, but the value for this attribute is different for each part.

For those readers familiar with object-oriented programming, attributes in Arena™ are similar in *concept* to attributes associated with objects in languages such as VB.Net, Java, and C++. If you understand the use of attributes in those languages, then you have a basis for how attributes can be used within Arena™. When an instance of an entity is created within Arena™, it is like creating an object in an object-oriented language. The object instance has attributes associated with it. You can think of an entity as a record or a row of data that is associated with that particular object instance. Each row represents a different entity.

Table 5.1 presents six entities conceptualized as records in a table. The column IDENT represents the IDENT attribute, which uniquely identifies each entity. IDENT is a special predefined attribute that Arena™ uses to uniquely track entities currently in the model. The IDENT attribute is assigned when the entity is created. When an entity is created within Arena™, memory is allocated for the entity and all of its attributes. A different value for IDENT is assigned to each entity that currently exists within the model. When an entity is disposed, the memory associated with that entity is released and the *values* for IDENT will be reused for newly created entities. Thus, the IDENT attribute uniquely identifies entities currently in the model. No two entities currently in the model have the same value for the IDENT attribute. You can think of the creating and disposing of entities as adding and deleting records within the model’s “entity table.” The *Entity.SerialNumber* attribute is also a unique number assigned to an entity when it is created; however, if the entity is ever duplicated (cloned) in the model, the clones will have the same value for the *Entity.SerialNumber* attribute. In the table, there are two entities (1 and 4) that are duplicates of each other. The duplication of entities will be discussed later in this chapter. The size, weight, and processing time attributes are three *user-defined* attributes associated with each of the entities. To find a description of the full list of predefined entity attributes, do a search on “Attributes and Entity-Related Variables” in the Arena™ Help system.

**TABLE 5.1 Entities Conceptualized as Records**

IDENT	Entity.SerialNumber	Size	Weight	Processing Time
1	1001	2	33	20
2	1002	3	22	55
3	1003	1	11	44
4	1001	2	33	20
5	1004	5	10	14
6	1005	4	14	10

**TABLE 5.2** Different Types of Entities Conceptualized as Records

IDENT	Type	Size	Weight	Move Time	Processing Time
1	1	2	33	—	20
2	1	3	22	—	55
3	2	1	11	23	—

When you define a user-defined attribute, you are adding another “column” to the “entity table.” The implication of this statement is very important. Defining a user-defined attribute associates that attribute with *every* entity type. This is quite different from what you see in object-oriented languages where you can associate specific attributes with specific classes of objects. Within Arena™, an entity is an entity. You can specify attributes that can allow you to conceptualize them as different types. This is conceptually defining the attribute set for the entity type that was previously mentioned. For example, you might create entities and think of them as parts flowing in a manufacturing system. Within a manufacturing system, there might also be entities that represent pallets that assist in material handling within the system. A part may have a processing time attribute and a pallet may have a move time attribute.

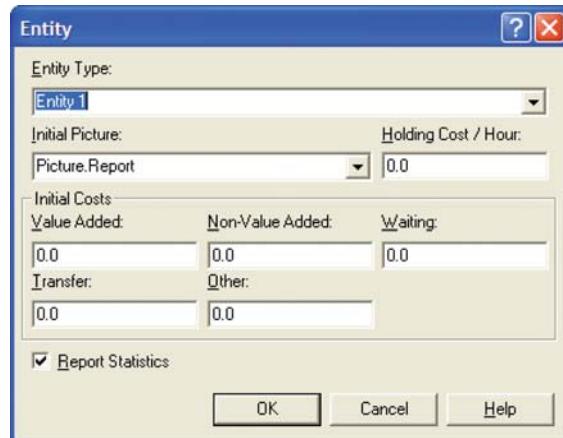
In Table 5.2, the type attribute indicates the type of entity (part = 1, pallet = 2). Notice that all entities have the move time and processing time attributes *defined*. These attributes can be used regardless of the actual type of entity. In using attributes within Arena™, it is up to the modeler to provide the appropriate context (understanding of the attribute set) for an entity and then to use the attributes appropriately within that context.

There are two ways to distinguish between different types of entities. Arena™ has a predefined attribute, called *Entity.Type*, which can be used to set the type of the entity. In addition, you can specify a user-defined attribute to indicate the type of the entity. Arena’s online help has this to say about the *Entity.Type* attribute:

*Entity.Type* This attribute refers to one of the types (or names) of entities defined in the Entities element. Entity type is used to set initial values for the entity picture and the cost attributes. It is also used for grouping entity statistics (e.g., each entity’s statistics will be reported as part of all statistics for entities of the same type).

The Entity module in the Basic process panel allows you to define entity types. The dialog box shown in Figure 5.10 shows the basic text fields for defining an entity type. The most important field is the “Entity Type” field, which gives the name of the entity type. The *Initial Picture* field allows a picture to be assigned to the *Entity.Picture* attribute when entities of this type are created. The other fields refer to how costs are tabulated for the entity when applying activity-based costing (ABC) functionality. This functionality will be discussed in Chapter 10.

The advantage of using the *Entity.Type* attribute is that Arena™ then makes other functions easier, such as collecting statistics by entity type (via the DISPOSE module), ABC, and displaying the entity picture. Using the *Entity.Type* attribute makes it a little more difficult to randomly assign an entity type to an entity, although this can still be done using the set concept, which will be discussed later. By using a user-defined attribute, you can change and interpret the attribute very easily by mapping a number to the type as was done in Table 5.2.



**Figure 5.10** Entity module dialog box.

Now if a basic understanding of the concept of an entity attribute has been developed, you still need to know how to declare and use the attributes. The declaration and use of attributes in Arena™ is similar to how variables can be declared and used in Visual Basic. Unless you have the *Option Explicit* keyword specified for Visual Basic, any variables can be declared upon their first use. For attributes in Arena™, this also occurs. For example, to make an attribute called *myArea* available, simply use it in a module, such as an ASSIGN module. This does cause difficulties in debugging. It is also recommended that you adopt a naming convention for your attributes. This text uses the convention of placing “my” in front of the name of the attribute, as in *myArea*. This little mnemonic indicates that the attribute belongs to the entity. In addition, spaces within the name are not used, and the convention of the concatenation and the capitalization of the words within the name is used. Of course, you are free to label your attributes whatever you desire as long as the names do not conflict with already existing attributes or keywords within Arena™. Attributes can also be formally declared using the ATTRIBUTES module. This also allows the user to define attributes that are arrays.

## 5.5 PROCESSING ENTITIES

This section overviews the types of operations facilitated by the Basic Process template panel and some of the programming constructs within Arena™. This is not meant to be a detailed discussion on these constructs, but rather a conceptual overview of the capabilities in order to set up the examples later in this and other chapters.

Entities move between modules via a direct connection or, as will be seen later, via an entity transfer mechanism. When an entity moves via a direct connection, the movement is instantaneous. That is, the entity completes the movement at the current time without any advance of the simulation clock. Some modules within Arena™ execute at the current event time, and others may cause the flow of the entity to stop its movement until the operation is completed. The entity that is currently moving is called the *active entity*. The active entity will continue to move until it is delayed or blocked by some module. To implement a simple time duration delay, you can use the PROCESS module on the Basic Process template with

the delay option selected or you can use the DELAY module from the Advanced Process template. This allows the modeling of entities that experience activities within the system.

An activity can take place with or without a resource. An activity that takes place without a resource is called unconstrained. For example, the walking of customers from one location to another within the grocery store can be modeled with an unconstrained activity (e.g., a process delay). Entities enter the delay, schedule an event representing the end of the activity, and then wait until the end of the activity event occurs before continuing their motion within the model. In some instances, the activity will require a resource. In which case, the entity must be allocated the required number of units of the resource before starting the activity. These types of activities are called resource constrained. For example, suppose that a part needs to move from a drilling station to a grinding station and that the part requires a fork-lift to be able to move. This type of movement is resource (fork-lift) constrained. Resource-constrained activities within a system are often modeled using the RESOURCE module within Arena™. Resource-constrained movement, as in the part/fork-lift example, can be modeled using either a RESOURCE or a TRANSPORTER module (found on the Advanced Transfer panel). When the entity's movement is constrained, the entity will need a place to wait. The QUEUE module defines the characteristics of the ordering of the waiting line. In most instances, if a module may constrain the movement of an entity, a queue will be automatically attached to the module. For example, the HOLD module allows the entity to be held in a queue until a specific signal is activated by another entity or until a specific condition is met.

In most practical situations, an entity may follow alternative paths through the system. In this case, the modeler needs to be able to direct the entity through the system. Two fundamental mechanisms are available for directing entities: probabilistic and conditional. In the case of probabilistic routing, the entity selects from a set of paths according to a random distribution. In the case of conditional routing, the entity selects a path based on a set of conditions within the system. Both of these mechanisms are available via the DECIDE module within Arena™. Besides the DECIDE module, Arena™ allows the selection of actions via various rules. For example, a resource can be selected based on the least number of busy units. Arena™ also has a number of programming-based mechanisms (e.g., IF-THEN-ELSE and WHILE) available within the BLOCKS template that facilitate entity flow. Finally, Arena™ has many advanced ways to define routes through the system via the Advanced Transfer panel. These include various material handling constructs such as conveyors and transporters.

Besides the CREATE module, the SEPARATE module allows entities to duplicate themselves within a model. As a complement to the SEPARATE module, the BATCH module allows a set of entities to be combined into either a permanent or a temporary group. In the case of a temporary group, the entities are represented as one entity that can be separated via the SEPARATE module back into the original entities. This allows the entities to move together as one unit (e.g., a pallet of items). Entities can also pickup and drop off other entities as they move via the PICKUP and DROPOFF modules on the Advanced Process panel. Finally, entities can wait in queues until a matching condition occurs via the MATCH module. While this does not group the entities, it allows their movements to be synchronized when the matching condition is met.

In the previously mentioned process modeling concepts, most of the concepts correspond to some physical phenomenon (e.g., using a resource and traveling from one location to another). These constructs assist in representing the physical system within the simulation model. However, these constructs must be augmented by programming constructs that

facilitate other aspects of simulation modeling. In particular, there must be ways to get input into the model and to get output from the model, to set up and run the model, to collect statistics, to assign variables/attributes, to animate the model, and to trace/debug the simulation model.

Input and output are available in a number of different formats (e.g., text files, spreadsheets, databases, and VBA forms). The basic way to handle input and output within an Arena™ model comes from the READWRITE module and the FILES module. In order to set up and run the model, the Run/Setup dialog can be used. To collect statistics within a module, the RECORD and STATISTIC modules are used. The ASSIGN module can be used to change the value of various attributes and variables within a model. Finally, to trace and debug a model, you can use some of the built in animation and also the Run Controller.

You should now have an overview of the basic concepts available for modeling within Arena™. The rest of this chapter and other chapters will examine the application of these concepts in a number of examples.

## 5.6 ATTRIBUTES, VARIABLES, AND SOME I/O

In this section, you are introduced to how to use attributes and variables. In addition, statistical collection and writing data to a file are introduced by expanding on the pharmacy model of Chapter 4.

### 5.6.1 Modifying the Pharmacy Model

Suppose that customers who arrive to the pharmacy have 1, 2, or 3 prescriptions to be filled. There is a 50% chance that they have 1 prescription to fill, a 30% chance that they have 2 prescriptions to fill, and a 20% chance that they have 3 prescriptions to fill. The time to service a customer is still exponentially distributed but the mean varies according to how many prescriptions that they need filled as shown in Table 5.3.

For illustrative purposes, current number of customers having 1, 2, or 3 prescriptions in system is needed within the animation of the model. In addition, the total number of prescriptions in the system should also be displayed. Statistics should be collected on the average number of prescriptions in the system and on the average time a customer spends in the system. For diagnostic purposes, the following information should be captured to a file:

- When a customer arrives: the current simulation time, the entity number (IDENT), the entity's serial number, the number of prescriptions for the customer;
- When a customer departs: the current simulation time, the entity number (IDENT), the entity's serial number, the number of prescriptions for the customer, the time of arrival, and the total time spent in the system.

**TABLE 5.3 Prescription-Related Data**

# Prescription	Chance	Service Time Mean, Minutes
1	50%	3
2	30%	4
3	20%	5

In addition to the above information, each of the above two cases should be denoted in the output. The simulation should be run for only 30 customers.

Until you get comfortable with your modeling skills, you should follow a basic modeling recipe and develop answers to the following questions:

- *What is the system? What information is known by the system?*
- *What are the required performance measures?*
- *What are the entities? What information must be recorded or remembered for each entity? How are entities introduced into the system?*
- *What are the resources that are used by the entities? Which entities use which resources and how?*
- *What are the process flows? Sketch the process or make an activity flow diagram.*
- *Develop pseudo-code for the situation.*
- *Implement the model in Arena<sup>TM</sup>.*

By considering each of these questions in turn, you should have a good start in modeling the system.

In this example, the system is again the pharmacy and its waiting line. The system knows about the chance for each prescription amount, the interarrival time distribution, and the service time distributions with their means by prescription amount. The system must also keep track of how many prescriptions of each type are in the pharmacy and the total number of prescriptions in the pharmacy. The performance of the system will be measured by using the average number of prescriptions in the system and the average time a customer spends in the system. As in the previous model, the entity will be the customers. Each customer must know the number of prescriptions needed. As before, each customer requires a pharmacist to have their prescription filled. The activity diagram will be exactly the same as in Chapter 4 because the activity of the customer has not changed.

When implementing this model within Arena<sup>TM</sup>, the following modules will be used:

- CREATE. This module will be used to create the 30 customers.
- ASSIGN. This module will be used to assign values to variables and attributes.
- READ/WRITE. This module will be used to write out the necessary information from the customers.
- PROCESS. This module will be used to implement the prescription filling activity with the required pharmacist.
- RECORD. This module will be used to record statistics on the system time of the customers.
- DISPOSE. This module will be used to dispose of the entities that were created.
- FILE. This data module defines the characteristics of the operating system file used by the READ/WRITE module.
- VARIABLE. This data module will be used to define variables to track the number of customers having the different amounts of prescriptions and to track the total number of prescriptions.

To start the modeling, the variables and attributes necessary for the problem need to be understood. In the problem, the system needs to keep track of the total number of prescriptions and the number of customers that need 1, 2, or 3 prescriptions filled. Therefore,

variables to keep track of each of these items are required. A scalar variable can be used to track the number of prescriptions (*vNumPrescriptions*) in the system and a 1D variable (*vNP*) with three elements can be used to track the number of customers requiring each of the three prescription amounts.

In the example, each customer needs to know how many prescriptions that he/she needs to have filled. Therefore, the number of prescriptions (*myNP*) needs to be an entity attribute. In addition, the arrival time of each customer and the customer's total time in the system must be written out to a file. Thus, each entity representing a customer must remember when it arrived to the pharmacy. Do real customers need to remember this information? No! However, the *modeling* requires this information. Thus, attributes (and variables) can be additional characteristics required by the modeler that are not necessarily part of the real system. Thus, the arrival time (*myArriveTime*) of each customer should be modeled as an entity attribute.

To understand why this information must be stored in an attribute consider what would happen if you tried to use a variable to store the number of prescriptions of the incoming customer. Entities move from one block to another during the simulation. In general, entities may delay their progress due to simulation logic. When this happens, other entities are allowed to move. If the number of prescriptions for the customer was stored in a (global) variable, then during the movement cycle of the other entities, the other entities might change the value of the variable. Thus, the information concerning the current customer's number of prescriptions will be over written by the next customer that enters the system. Thus, this information must be carried with the entity. Global variables are very useful for communicating information between entities; however, as in any programming language, care must be taken to use them correctly.

The information in Table 5.3 also needs to be modeled. Does this information belong to the system or to each customer? While the information is about customers, each customer does not need to "carry" this information. The information is really about how to create the customers. The system must know how to create the customers; thus this information belongs to the system. The number of prescriptions per customer can be modeled as a random variable with a discrete distribution. The information about the mean of the service time distributions can be kept in a 1D variable with three elements representing the mean of each distribution. The basic pseudo-code for this situation is given in Exhibit 5.1.

In what follows, the previously developed pharmacy model will be used as a basis for making the necessary changes for this situation. If you want to follow along with the construction of the model, then make a copy of the pharmacy model of Chapter 4. You should start by defining the variables for the example. With your copy of the model open, go to the Basic Panel, select the VARIABLE data module, and define the following variables:

- *vNumPrescriptions*. This scalar variable should count the total number of prescriptions in the system. This variable should be initialized at the value 0.0.
- *vNP*. This 1D variable should have three rows and should count the number of customers having 1, 2, and 3 prescriptions currently in the system. The index into the array is the prescription quantity. Each element of this variable should be initialized at the value 0.0.
- *vServiceMean*. This 1D variable should have three rows representing the mean of the service distributions for the three prescription quantities. The index into the array is the prescription quantity. The elements (1, 2, and 3) of this variable should be initialized at the values 3, 4, and 5, respectively.

---

**Exhibit 5.1** Multiple Prescription Pharmacy Model
 

---

```

CREATE customers with EXPO(6) time between arrivals
ASSIGN
  myNP = DISC(0.5, 1, 0.8, 2, 1.0, 3)
  vNumPrescriptions = vNumPrescriptions + myNP
  vNP(myNP) = vNP(myNP) + 1
  myArriveTime = TNOW
END ASSIGN
WRITE customer arrival information to a file
PROCESS customers
  SEIZE 1 pharmacist
  DELAY for EXPO(vServiceMean(myNP)) minutes
  RELEASE 1 pharmacist
END PROCESS
ASSIGN
  vNumPrescriptions = vNumPrescriptions - myNP
  vNP(myNP) = vNP(myNP) - 1
END ASSIGN
RECORD (TNOW - myArriveTime)
WRITE customer departure information to a file
DISPOSE customer
  
```

---

	Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics
1	vNumPrescriptions			Real	System	0 rows	<input checked="" type="checkbox"/>
2	vNP	3		Real	System	0 rows	<input type="checkbox"/>
3	vServiceMean	3		Real	System	3 rows	<input type="checkbox"/>
4	vEventType			Real	System	0 rows	<input type="checkbox"/>

Double-click here to add a new row.

Initial Values X

1	3.0
2	4.0
3	5.0

**Click on initial values button to edit initial values**

**Causes time based statistics to be collected**

**Figure 5.11** Defining the variables.

Given the information concerning the variables, the variables can be defined as shown in Figure 5.11. For scalar variables, you can tell Arena™ to automatically report statistics on the time-average value of the variable. Chapter 7 will more formally define time-average statistics, but for now, you can think of them as computing the average value of the variable over time. The initial values can be easily defined using the spreadsheet view.

Now, you are ready to modify the modules from the previous model within the model window. The CREATE module is already in the model. In what follows, you will be inserting some modules between the CREATE module and the PROCESS module and inserting some modules between the PROCESS module and the DISPOSE module. You should select and delete the connector links between these modules. You should then drag an ASSIGN module from the Basic Process panel to the model window so that you can connect it to the CREATE module.

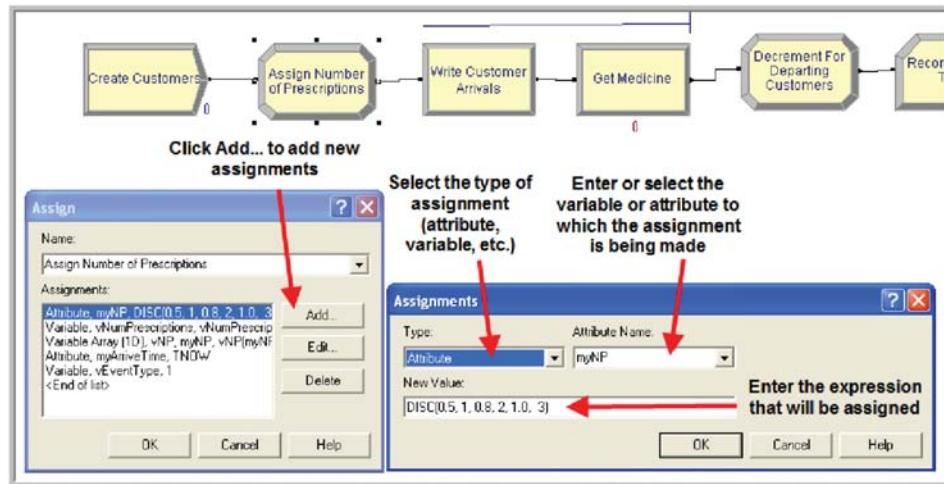


Figure 5.12 Assigning the number of prescriptions.

### 5.6.2 Using the ASSIGN Module

Because the ASSIGN module is directly connected to the CREATE module, each entity that is created will pass through the ASSIGN module causing each assignment statement to be executed in the order in which the assignments are listed in the module. The ASSIGN module simply represents a series of logical assignment, for example. Let  $X = 2$ , as you would have in any common programming language. Figure 5.12 illustrates the dialog boxes associated with the ASSIGN module. Using the Add button, you can add a new assignment to the list of assignments for the module. Add an attribute named “myNP” and assign it the value returned from  $\text{DISC}(0.5, 1, 0.8, 2, 1, 0, 3)$ .

The function  $\text{DISC}(cp_1, v_1, cp_2, v_2, \dots, cp_n, v_n)$  will supply a random number according to a discrete probability function, where  $cp_i$  are cumulative probabilities and  $v_i$  is the value, that is,  $P(X = v_i) = cp_{i+1} - cp_i$  with  $cp_0 = 0$  and  $cp_n = 1$ . The last cumulative value must be 1.0 since it must define a probability distribution. For this situation, there is a 50% chance of having 1 prescription, a  $(80 - 50 = 30\%)$  chance for 2 prescriptions, and  $(100 - 80 = 20\%)$  chance for 3 prescriptions.

The Expression Builder can be used to build complicated expressions by right-clicking in the text field and choosing Build Expression. Figure 5.13 illustrates the Expression Builder with the DISC distribution function and shows a list of available probability distributions. Once the number of prescriptions has been assigned to the incoming customer, you can update the variables that keep track of the number of customers with the given number of prescriptions and the total number of prescriptions. As an alternative to the dialog view of the ASSIGN module, you can use the spreadsheet view as illustrated in Figure 5.14.

You should complete your ASSIGN module as shown in Figure 5.14. The spreadsheet view of Figure 5.14 clearly shows the order of the assignments within the ASSIGN module. The order of the assignments is important! The first assign is to the attribute *myNP*. This attribute will have a value 1, 2, or 3 depending on the random draw caused by the DISC function. This value represents the current customer’s number of prescriptions and is used in the second assignment to increment the variable that represents the total number of prescriptions in the system. In the third assignment, the attribute *myNP* is used to index into the variable array that counts the number of customers in the system with the given number of prescriptions. The fourth assignment uses the Arena™ variable TNOW to assign the

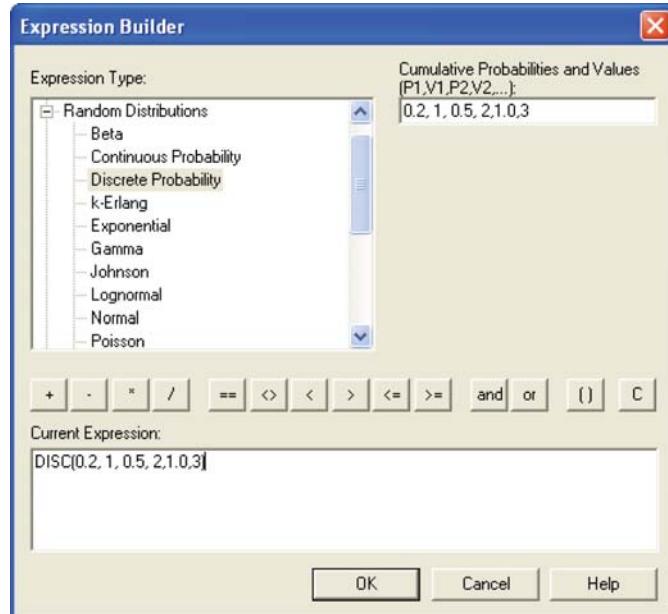


Figure 5.13 Expression Builder.

Assignments					
	Type	Variable Name	Row	Attribute Name	New Value
1	Attribute	Variable 1	1	myNP	DISC(0.5, 1, 0.8, 2, 1, 0, 3)
2	Variable	vNumPrescriptions	1	Attribute 2	vNumPrescriptions + myNP
3	Variable Array (1D)	vNP	myNP	Attribute 3	vNP(myNP) + 1
4	Attribute	Variable 4	1	myArriveTime	TNOW
5	Variable	vEventType	1	Attribute 5	1

Double-click here to add a new row.

Assign - Basic Process

	Name	Assignment
1	Assign Number of Prescriptions	5 rows
2	Decrement For Departing Customers	3 rows

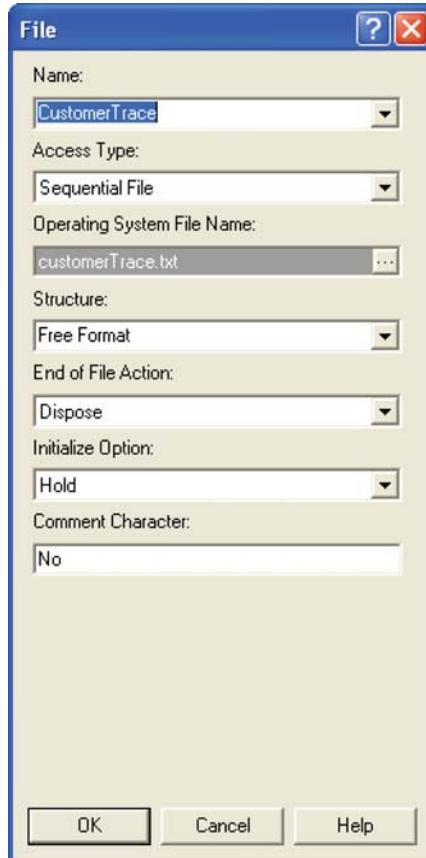
**Click to see spreadsheet view of ASSIGN module**

Figure 5.14 Spreadsheet view of ASSIGN module.

current simulation time to the attribute *myArriveTime*. TNOW holds the current simulation time. Thus, the attribute *myArriveTime* will contain the value associated with TNOW when the customer arrived, so that when the customer departs the total elapsed time in the system can be computed. The last assignment is to a variable that will be used to denote the type of event 1 for arrival, 2 for departure when writing out the customer's information.

### 5.6.3 Using the READWRITE Module

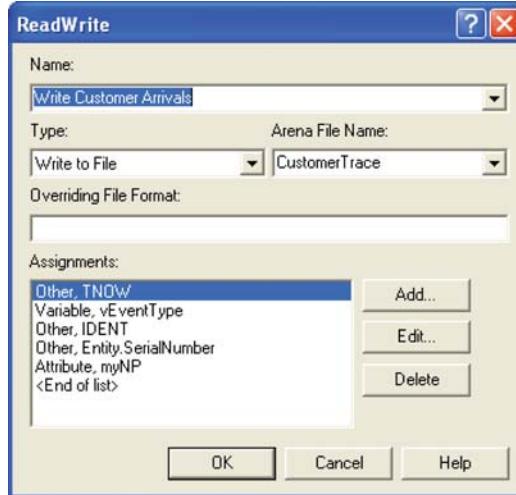
As with any programming language, Arena™ has the ability to read and write information from files. Often the READWRITE module is left to the coverage of miscellaneous topics in Arena™; however, in this section, the READWRITE module is introduced so that you can be aware that it is available and possibly use it when debugging your models. The



**Figure 5.15** The FILE module.

READWRITE module is very useful for writing information from a simulation and for capturing values from the simulation, for post processing, and other analysis.

This example uses the READWRITE module to write out the arriving and departing customer's information to a text file. The information can be used to trace the customer's actions through time. There are better ways to trace entities with Arena<sup>TM</sup>, but this example uses the READWRITE module so that you can gain an understanding of how Arena<sup>TM</sup> processes entities. In order to use the READWRITE module, you must first define the file to which the data will be written. You do this by using the FILE data module in the Advanced Process panel. To get the FILE dialog, insert a row into the file area and then choose edit via dialog from the right-click context menu. Open the FILE dialog and make it look like FILE module shown in Figure 5.15. This will define a file within the operating system to which data can be written or from which data can be read. Arena's FILE module allows the user to work with text files, Excel files, Access files, as well as the other access types available in the *Access Type* drop down menu dialog. In this case, the *Sequential* option has been chosen, which indicates to Arena<sup>TM</sup> that the file will be a text file for which the records will be in the same sequence that they were written. In addition, the *Free Format* option has been selected, which essentially writes each value to the file separated by a space.



**Figure 5.16** The READWRITE module.

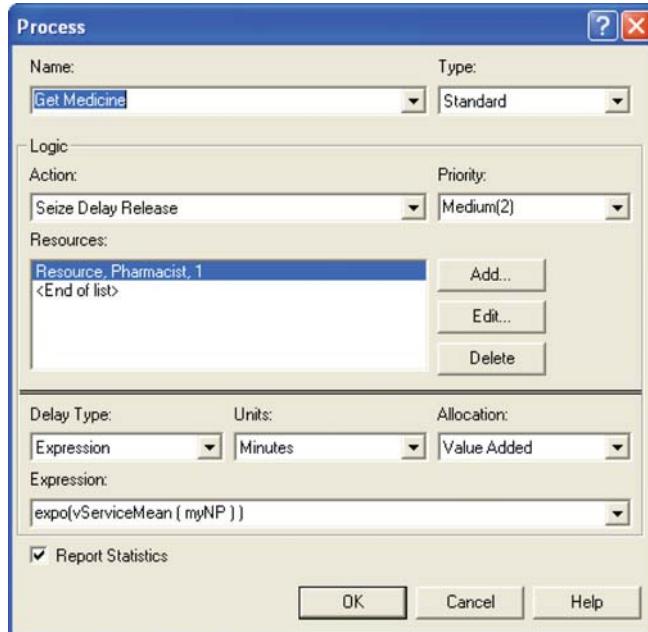
You should review the help file on the FILE module for additional information on the other access types.

Now, open the READWRITE module and make it look like the READ/WRITE module given in Figure 5.16. Use the drop down menu to select the file that you just defined and use the *Add* button to tell the module what data to write out. After selecting the Add button, you should use the *Type* drop down to select the data type of the item that you want to write out to the file and the item to write out. For the data type Other, you must type in the name of the item to be written out (it will not be available in a drop down context). In this case, the following is being written to the file:

- *TNOW*, the current simulation time,
- *vEventType*, the type of event 1 for arrival, 2 for departure,
- *IDENT*, the entity identity number of the customer,
- *Entity.SerialNumber*, the serial number of the customer entity,
- *myNP*, the number of prescriptions for the customer.

This information is primarily about the current (active) entity. The active entity is the entity that is currently moving through the model's modules. In addition to writing out information about the active entity or the state of the system, the READ/WRITE module is quite useful for reading in parameter values at the beginning of the simulation and for writing out statistical values at the end of the simulation. These options will be explored in Chapter 10 when illustrating how to read/write to Excel and Access.

After writing out the values to the file, the customer entity proceeds to the PROCESS module. Because the mean of the service time distribution depends on the number of prescriptions, we need to change the PROCESS module. In Figure 5.17, we use the 1D array, *vServiceMean*, and the attribute, *myNP*, to select the appropriate mean for the exponential distribution in the PROCESS module. Remember that the attribute *myNP* has a value 1, 2, or 3 depending on the number of prescriptions for the customer. This value is used as the



**Figure 5.17** Changed PROCESS module.

index into the *vServiceMean* 1D variable to select the appropriate mean. Since Arena<sup>TM</sup> has limited data structures, the ability to use arrays in this manner is quite common.

Now, if the customer has received service, you can update the variables that keep track of the number of prescriptions in the system and the number of customers having 1, 2, and 3 prescriptions. This can be done with an ASSIGN module placed after the PROCESS module. Figure 5.18 shows the assignments for after a customer completes service. The first assignment decrements the global variable, *vNumPrescriptions*, by the amount of prescriptions, *myNP*, denoted by the current customer. Then, this same attribute is used to index into the array that is counting the number of customers that have 1, 2, or 3 prescriptions to reduce the number by 1 since a customer of the designated type is leaving. In preparation for writing the departing customer's information to the file, you should set the variable *vEventType* to 2 to designate a departure.

#### 5.6.4 Using the RECORD Module

The problem states that the average time in the system must be computed. To do this, the RECORD module can be used. The RECORD module is found on the Basic Process panel. The RECORD module tabulates information each time an entity passes through it. The options include the following:

- *Count* will increase or decrease the value of the named counter by the specified value.
- *Entity Statistics* will generate general entity statistics, such as time and costing/duration information.
- *Time Interval* will calculate and record the difference between a specified attribute's value and the current simulation time.

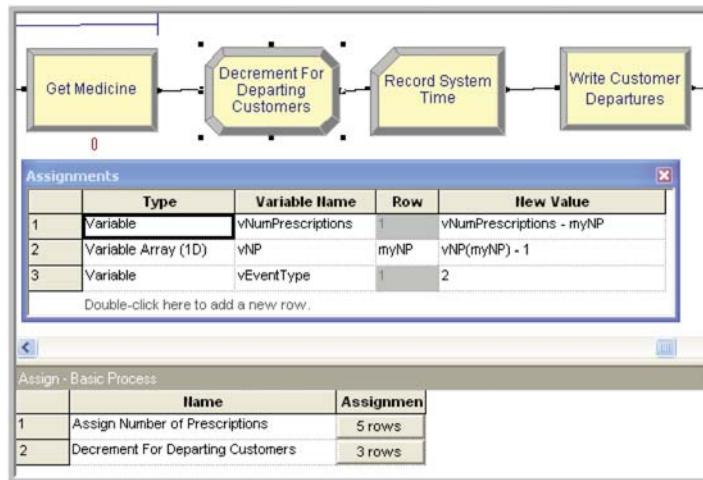


Figure 5.18 Updating variables after service.

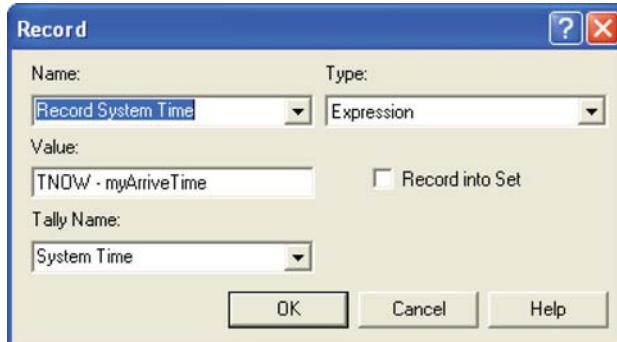
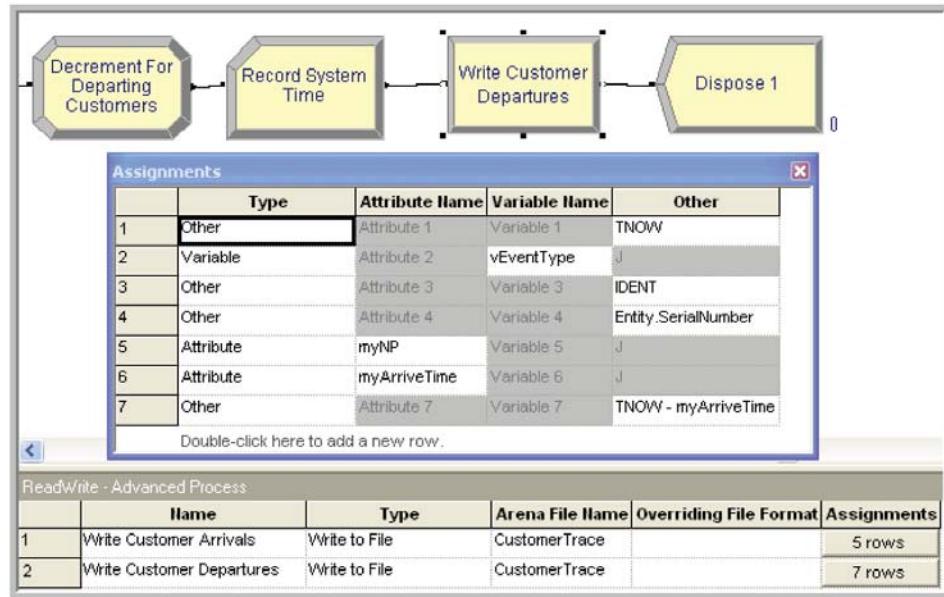


Figure 5.19 The RECORD module.

- *Time Between* will track and record the time between entities entering the module.
- *Expression* will record the value of the specified expression.

Drag and drop the RECORD module after the previous ASSIGN module. Open the record module and make it look like the RECORD module shown in Figure 5.19. Make sure to choose the Expression option. The RECORD module evaluates the expression given in the Value text box field and passes the value to an internal Arena™ function that automatically tallies statistics (min, max, average, standard deviation, count) on the value passed. Since the elapsed time in system is desired, the current time minus when the customer arrived must be computed. The results of the statistics will be shown on the default Arena™ reports labeled with the name of the tally, for example, *SystemTime*. The Time Interval option could also have been used by supplying the *myArriveTime* attribute. You should explore and try out this option to see that the two options produce the same results.

The final model change is to include another READWRITE module to write out the information about the departing customer. The READWRITE module also has a spreadsheet view for assigning the information that is to be read in or written out.



**Figure 5.20** READWRITE module for departing customers.

Figure 5.20 shows the spreadsheet view for the departing customer's READWRITE module.

The module is set to first write out the simulation time via the variable TNOW, then the type of event, *vEventType*, which was set to 2 in the previous ASSIGN module. Then, it writes out the information about the departing entity (IDENT, Entity.SerialNumber, myNP, myArriveTime). Finally, the system time for the customer is written out to the file.

### 5.6.5 Animating a Variable

The problem asks to have the value of the number of prescriptions currently in the system displayed in the model window. This can be done by using the variable animation features in Arena™. Figure 5.21 shows the animation toolbar within Arena™. If the toolbar is not showing in your Arena™ environment, then you can right-click in the toolbar area and make sure that the Animate option is checked as shown in Figure 5.21.

You will use the Animate Variables option of the animate toolbar to display the current number of prescriptions in the system. Click on the button with the 0.0 on the toolbar. The animate variable dialog will appear. See Figure 5.22. By using the drop down box labeled Expression, you can select the appropriate item to display. After filling out the dialog box and selecting OK, your cursor will change from the arrow selection form to the cross-hairs form. By placing your cursor in the model window and clicking, you can indicate where you want the animation to appear within the model window. After the first click, drag over the area where you want the animation to size the animation and then click again. After completing this process, you should have something that looks like Figure 5.23. During the simulation run, this area will display the variable *vNumPrescriptions* as it changes with respect to time.

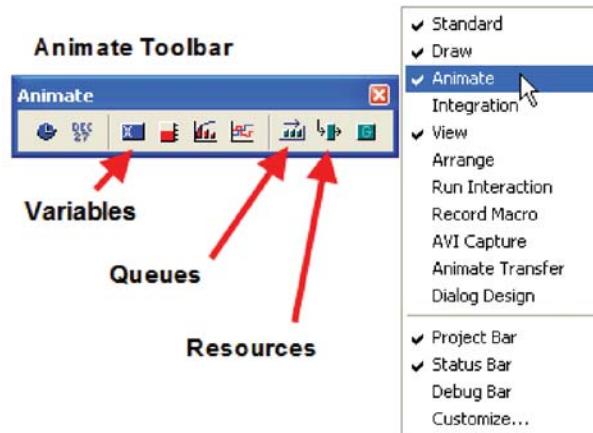


Figure 5.21 The animate toolbar.

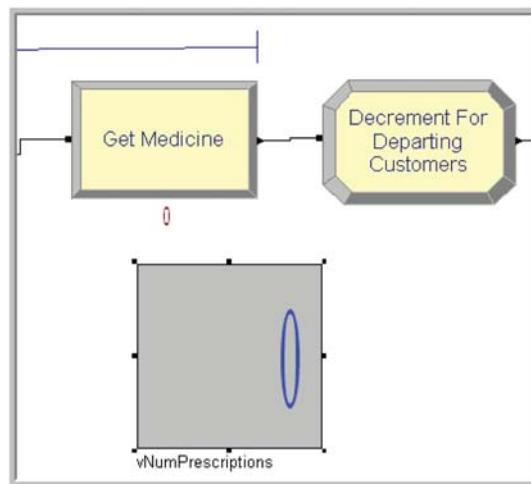


Figure 5.22 Animate variable dialog.

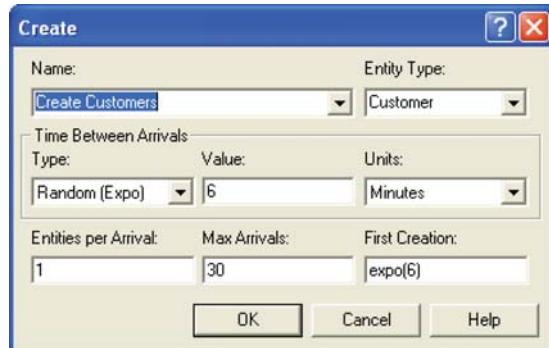
### 5.6.6 Running the Model

Two final changes to the model are necessary to prepare the model to run only 30 customers. First, the maximum number of arrivals for the CREATE module must be set to 30. This will ensure that only 30 customers are created by the CREATE module. Second, the Run Setup > Replication Parameters dialog needs to indicate an infinite replication length.

These two changes (shown in Figures 5.24 and 5.25) ensure that the simulation will stop after 30 customers are created. If you do not specify a replication length for the simulation, Arena™ will process whatever entities are created within the model until there are no



**Figure 5.23** Animating the number of prescriptions in the system.



**Figure 5.24** Creating only 30 customers.

more entities to process. If a maximum number of arrivals for the CREATE module is not specified and the replication length was infinite, then the simulation would never end.

After completing all these changes to the model, use Run Setup > Check Model to check if there are any syntax errors in your model. If there are, you should carefully go back through the dialog boxes to make sure that you typed and specified everything correctly. The final Arena™ file *PharmacyModelEx2-1.doe* is available in the supporting files for this chapter. To see the animation, you should make sure that you unchecked the Batch Run (No animation) option in the Run > Run Control menu. You should run your model and agree to see the simulation report. When you do, you should see that 30 entities passed through the system. By clicking on the user-specified area of the report, you will see the statistics collected for the RECORD module and the VARIABLE module that was specified when constructing the model.

Figure 5.26 indicates that the average system time of the 30 customers who departed the system was about 7.1096. Since the system time was written out for each departing customer, Arena's calculations can be double checked. After running the model, there should

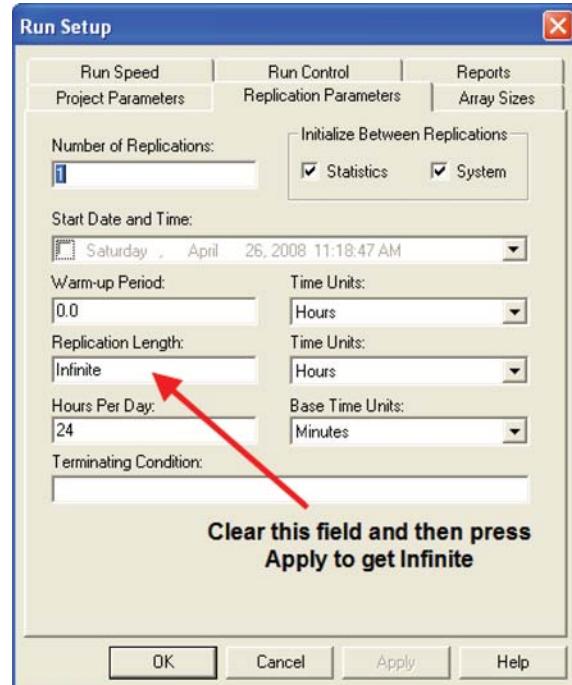


Figure 5.25 No replication length specified.

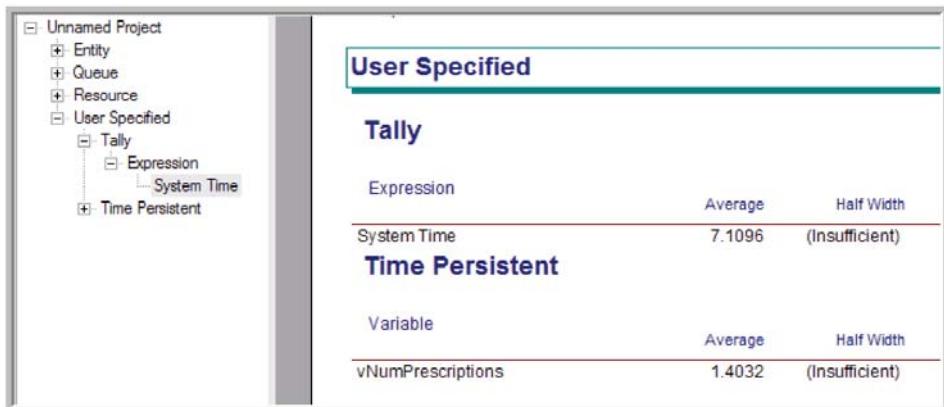


Figure 5.26 System time output report.

be a text file called *customerTrace.txt* in the same directory as where you ran the model. Table 5.4 shows the data from the first five customers through the system.

The table indicates that the first customer arrived at time 2.076914 with 1 prescription and was given the IDENT value 2 and serial number 1. The customer then departed, event type = 2, at time 2.257428 with a total system time of  $(2.257428 - 2.076914 = 0.180514)$ . There were then three consecutive arrivals before the second customer (IDENT = 3, serial number 2) departed at time 13.31427. Thus, the third and fourth customers had to wait in

**TABLE 5.4 Data from Output File in Tabular Form**

TNOW	Event Type	IDENT	Serial Number	Number Prescriptions	Arrive Time	System Time
2.076914	1	2	1	1		
2.257428	2	2	1	1	2.076914	0.180514
4.74824	1	3	2	1		
5.592303	1	4	3	3		
10.34295	1	5	4	1		
13.31427	2	3	2	1	4.74824	8.566026
21.39914	1	6	5	1		
21.89796	2	4	3	3	5.592303	16.30566
23.79078	2	5	4	1	10.342949	13.447826
26.0986	2	6	5	1	21.399139	4.699463

the queue. If you compute the average of the values of the system time column in the file, you would get the same value as that computed by Arena™ on the reports.

In this example, you have learned more about the CREATE module and you have seen how you can define variables and attributes within your models. With the use of the ASSIGN module, you can easily change the values of variables and attributes as entities move through the model. In addition, the RECORD module allows you to target specific quantities within the model for statistical analysis. Finally, the READWRITE module was used to write out information concerning the state of the system and the current entity when specific events occurred within the model. This is useful when diagnosing problems with the model or for capturing specific values to files for further analysis. The simple variable animation feature of Arena™ was also demonstrated.

This example had a simple linear flow, with each module directly connected to the previous module. Complex systems often have more complicated flow patterns. The next section introduces how to better control the flow of entities through the model.

## 5.7 FLOW OF CONTROL IN ARENA

When developing computer programs in standard languages, programmers have the ability to direct the flow of the program so that certain statements are executed, perhaps repeatedly. In Arena™, the flowchart programming paradigm makes this flow of control somewhat obvious. The entity follows the connected flowchart symbols. In the examples that have been considered so far, the flow has been linear from one connected module to another.

There are three primary mechanisms for directing entities within the model:

1. Logical and probabilistic conditions
2. Iterative looping
3. Entity transfers.

The first two methods are based on connecting the flowchart symbols and directing the entity along the linked symbols. The last method is related to modeling entity movement and material handling constructs within a model. Entity transfers will be discussed in Chapter 9.

### 5.7.1 Logical and Probabilistic Conditions

A programming language such as C has constructs like *if-then-else* which can direct the execution of the program, given specific conditions. Arena™ has three constructs for modeling conditional logical tests, DECIDE, BRANCH, and IF-ELSE-ENDIF. The DECIDE module is the more commonly used form and is found on the Basic Process template panel. Figure 5.27 illustrates the DECIDE module flowchart symbol and its dialog box. As can be seen in the figure, a DECIDE module has four types (2-way by chance, 2-way by condition, N-way by chance, and N-way by condition).

The *by condition* options will be discussed first. The 2-way by condition option implements a basic if-then-else construct, with the entity being directed along either of the two indicated paths based on whether or not the condition is true or false. In pseudo-code form, the 2-way by condition DECIDE module acts as follows:

```
IF condition is true THEN
    send entity out the true exit point
ELSE
    send entity out the false exit point
END IF
```

Figure 5.28 illustrates the DECIDE module dialog for the 2-way by condition situation.

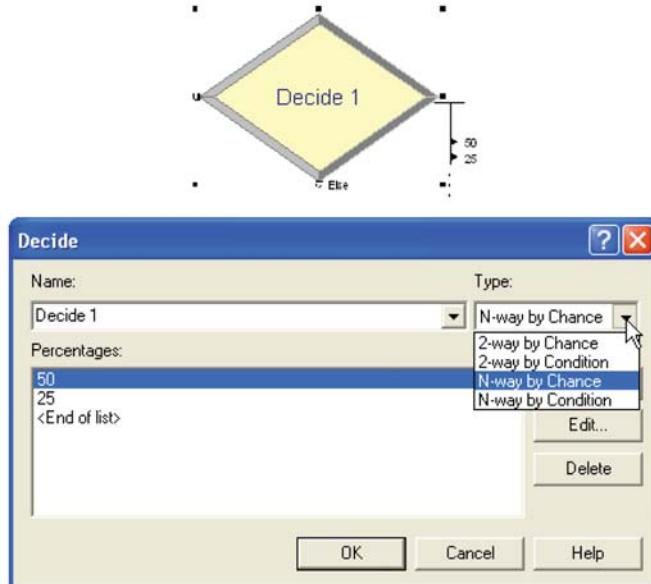
The N-way by condition option implements something similar to a switch statement in C or a case statement in Visual Basic. In pseudo-code form, the N-way by condition DECIDE module acts as follows:

```
IF condition 1 is true THEN
    send entity out the condition 1 exit point
ELSE IF condition 2 is true THEN
    send entity out the condition 2 exit point
ELSE IF condition 3 is true THEN
    send entity out the condition 3 exit point:
ELSE IF condition n is true THEN
    send entity out the condition n exit point
ELSE
    send entity out false exit point
END IF
```

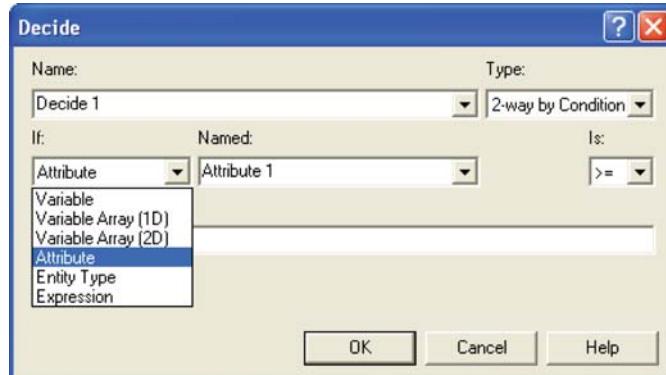
As soon as a condition tests to true, the entity is sent out through the exit point. No further tests are performed. The order of the tests is clearly important. By placing certain conditions before other conditions, the modeler can provide a simplistic priority over the conditions.

Conditions can be any legal Arena™ expression that evaluates to a logical (Boolean) value. The DECIDE module's dialog box facilitates the building of conditions. The user can write expressions for testing attributes, variables, array elements, and entity types. In addition, the user can write a general expression. The use of the expression builder is very useful in assisting with the development of expressions and helps with preventing syntax and spelling mistakes.

Table C.2 lists the mathematical and logical operators that can be used in Arena™ expressions. The logical operators have two forms. For example, equality can be tested either by using .EQ. or by using “= =.” The precedence for the operators is also listed in the table. Parenthesis can be used to group the sections of an expression so that the desired evaluation will be performed in the appropriate order.

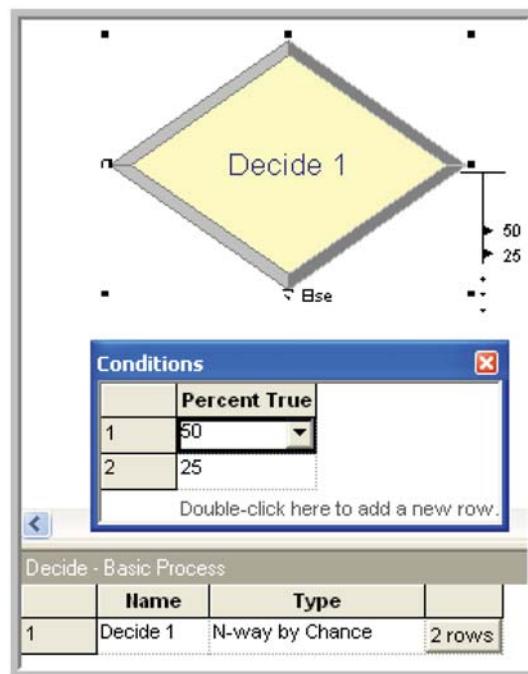


**Figure 5.27** DECIDE flowchart symbol and module dialog.



**Figure 5.28** The 2-way DECIDE module dialog box.

The *by chance* option for the DECIDE module allows the entity to randomly pick from the list of exit points. In the 2-way by chance option, the user specifies the probability associated with the true exit point. The remaining probability is associated with the false exit point. In the N-way by chance option (see Figure 5.29), the user specifies the percent true for each of the exit points. The *Else* exit point uses the remaining probability for its chance. The percent true for each exit point can be easily specified using the spreadsheet view as shown in Figure 5.29. When an entity enters the by chance DECIDE module, Arena<sup>TM</sup> randomly generates a number and selects the exit point based on the specified probability distribution across the exit points.

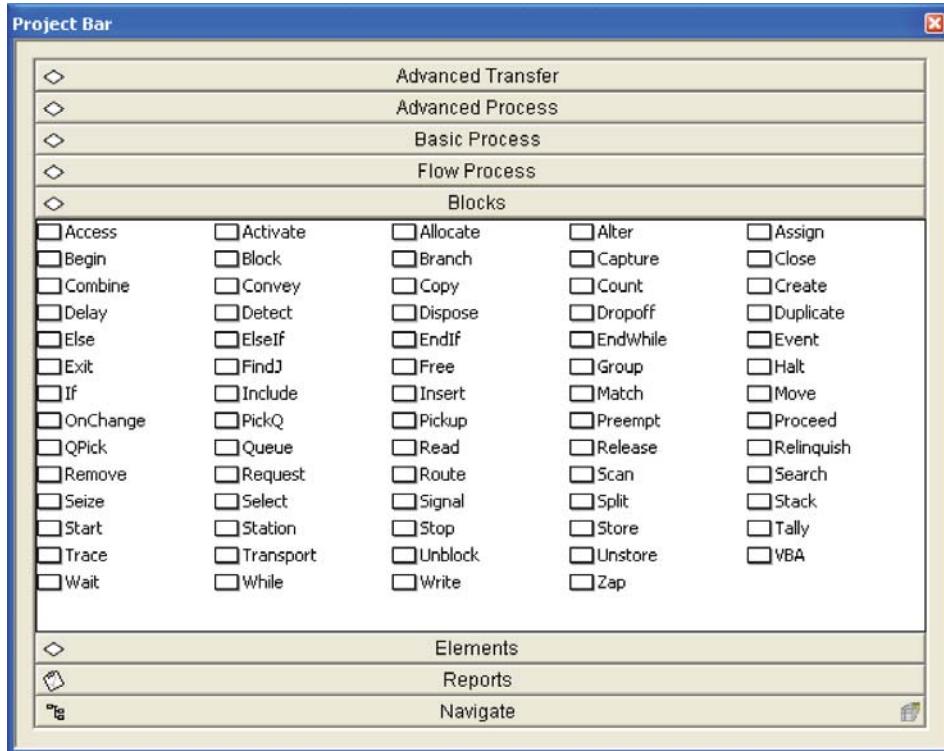


**Figure 5.29** N-Way chance DECIDE module.

The BRANCH block found on the Blocks template panel also performs very similarly to the DECIDE module. The Blocks Template contains the original SIMAN commands from which the other templates are derived, see Figure 5.30. SIMAN is the underlying text-based programming language for Arena™. SIMAN's relationship with Arena™ is discussed further later in this chapter. The Blocks template panel can be attached by right-clicking in the project bar and choosing *Blocks.tpo* from Arena's template folder. The BRANCH block works in a manner similar to the DECIDE module except that you specify the maximum number of branches, the branch type, and the random number stream. The use of random number streams is discussed in the next chapter.

The BRANCH block directs the arriving entity along one of the branches according to the type of branching. The key difference is the use of the maximum number of branches. If the maximum number of branches is 1, then this block works essentially the same as the DECIDE module; however, if the maximum number of branches is greater than 1, duplicate entities will be created and allowed to flow along the branches for which the conditions are met. The BRANCH block has been largely superseded by the DECIDE module. Unless you are certain that you need to utilize this duplicating entity functionality, you should stick with the DECIDE module. Further details on how the BRANCH block handles duplicates and the order that they exit the block are available within the Arena™ Help system by selecting Help on the BRANCH block dialog box.

The final logical flow construct that will be discussed is Arena's IF-ELSEIF-ELSE-ENDIF blocks. These blocks are also found on the Blocks template. These blocks work in a similar manner as the standard if-then-else constructs available in major languages.



**Figure 5.30** Blocks template panel.

Listing 5.1 shows an example for the use of the IF-ELSEIF-ELSE-ENDIF block combination from Arena’s help files. If the logical condition associated with the IF block is true, the entity will only execute the statements between the testing block and the next ELSEIF, ELSE, or ENDIF block combination. An ENDIF block must be used to terminate the use of combinations of these blocks. When using these blocks, the sets of statements are mutually exclusive. Only one set will execute based on the evaluation of the conditions tested. For example, in the exhibit, if PartType equals 1, then only the following statements will be executed:

```
SEIZE: Operator;
DELAY: ProcessTime;
RELEASE: Operator;
```

The use of these blocks requires the user to place and connect each of the modules, i.e. you cannot write this sort of text directly within the Arena™ environment. This often becomes tedious and takes up much screen real estate. The use of multiple DECIDE modules can largely take the place of the use of IF-ELSEIF-ELSE-ENDIF blocks; however, their use can be convenient for complicated model logic and if you prefer a more structured programming style.

**Listing 5.1 IF-ELSEIF-ELSE-ENDIF Example from Arena's Help File**

An entity arriving to this IF block proceeds to the next block in the model if there are any busy units of resource Sweeper. Otherwise, the entity is sent to the next ELSEIF, ELSE, or ENDIF block.

## IF-ELSEIF-ELSE-ENDIF Blocks

```

IF: PartType == 1;
    SEIZE: Operator;
    DELAY: ProcessTime;
    RELEASE: Operator;
ELSEIF: PartType == 14;
    DELAY: PreProcess;
    SEIZE: SpecialCrane;
    DELAY: MoveTime;
    RELEASE: SpecialCrane;
ELSE;
    DELAY: NormalProcess;
ENDIF;
```

Entities executing the above blocks are checked to see if PartType is equal to 1, in which case the entity seizes the resource Operator, delays for ProcessTime, and releases the Operator. If PartType is not equal to 1, a check is made to see if PartType is equal to 14, in which case the entity delays for PreProcess, seizes the resource SpecialCrane, delays for MoveTime, and releases SpecialCrane. If PartType is neither 1 nor 14, the entity delays for NormalProcess.

### 5.7.2 Iterative Looping

Iterative looping is often performed in computer programming. Languages such as C have constructs like for-loop, do-while, etc. to allow a set of statements to be iteratively executed. Arena™ has the WHILE/ENDWHILE blocks to perform this task as well as flowchart mechanisms to accomplish iteration, essentially by sending the entity back through a series of statements. The use of the WHILE-ENDWHILE combination is similar to that for the IF-ELSEIF-ELSE-ENDIF blocks. Exhibit 5.2 illustrates the use of the blocks in SIMAN code. Again, this text format cannot be directly developed within the Arena™ environment. The user must drag and drop and hook up the blocks within the model. An error will occur if a WHILE block is not matched up with a corresponding ENDWHILE block. You will examine iterative looping in Arena™ through the use of an example. In addition, you will learn more about variables, attributes, and a new construct call EXPRESSIONS.

**Listing 5.2 WHILE-ENDWHILE Arena Help Example**

```

Syntax
WHILE: Condition or Expression;

Basic Use
ASSIGN: J=0;
WHILE: J<10;
    ASSIGN: J = J + 1;
    ProcessTime(J)=PTime(1,J);
ENDWHILE;

Prompt          Entry
Condition or Expression J < 10

```

The entity first assigns J equal to 0. Then, while J is less than 10, the entity assigns ProcessTime(J) equal to PTime(1,J). This effectively assigns ProcessTime(1) through ProcessTime(10) equal to PTime(1,1) through PTime(1,10). As soon as J is incremented to 10, the loop is terminated and the entity is transferred to the ENDWHILE block.

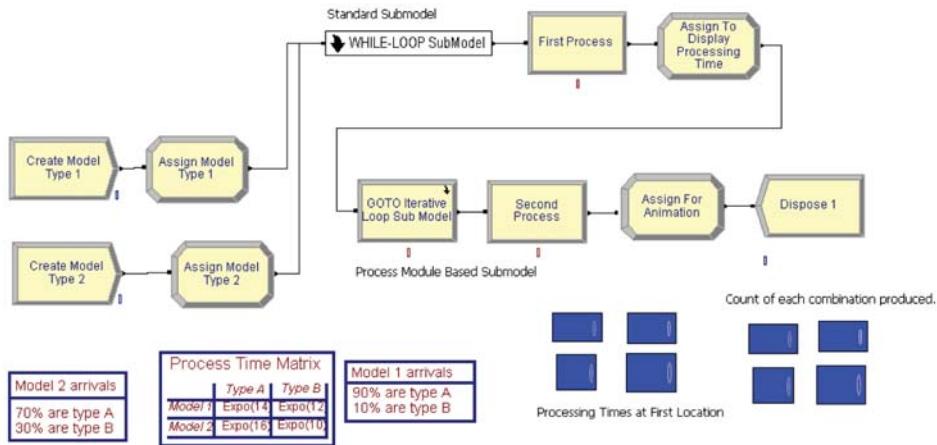
### 5.7.3 Example: Iterative Looping, Expressions, and Submodels

In this example, you will examine how to define and use attributes, variables, and expressions within Arena™. Also, you will learn how to delay an entity along its flow path. In addition, you will learn some basic flow of control mechanisms for looping or iterating within Arena™. This model will use the following modules:

- CREATE. Two instances of this module will be used to have two different arrival processes into the model.
- ASSIGN. This module will be used to assign values to variables and attributes.
- WHILE & ENDWHILE. These modules will be used to loop the entities until a condition is true.
- DECIDE. This module will also be used to loop entities until a condition is true.
- PROCESS. This module will be used to simulate simple time delays in the model.
- DISPOSE. This module will be used to dispose of the entities that were created.
- VARIABLES. This data module will be used to define variables and arrays for the model.
- EXPRESSIONS. This data module will be used to define named expression to be used within the model.
- Submodels are areas of the model window that contain modules that have been aggregated into one module.

This example is based partly on Arena's SMARTS file 183. This model will be considerably larger than the previous models. The final model will look something like that shown in Figure 5.31.

This system produces products. The products have different model configurations (model 1 and 2) that are being produced within a small manufacturing system. Model 1



**Figure 5.31** Completed model for iterative looping example.

arrives according to a Poisson process with a mean rate of 1 model every 12 minutes. The second model type also arrives according to a Poisson arrival process but with a mean arrival rate of 1 arrival every 22 minutes. Furthermore, within a model configuration, there are two types of products produced, type A and type B.

Table 5.5 shows the data for this example. In the table, 90% of model configuration 1 is of type A. In addition, the base process time for type A's for model configuration 1 is exponentially distributed with a mean of 14 minutes. Clearly, the base process time for the products depends on the model configuration and the type of product A or B. The actual processing time is the sum of 10 random draws from the base distribution. Processing occurs at two identical sequential locations. After the processing is complete, the product leaves the system.

When building this model, you will display the final processing time for each product at each of the two locations and display a count of the number of each configuration and type that was produced.

**5.7.3.1 Building the Model** Following the basic modeling recipe, consider the question: *What are the entities?* By definition, entities are things that flow through the system. They are transient. They are created and disposed. For this situation, the products are produced by the system. The products flow through the system. These are definite candidates for entities. In this problem, there are actually four types of entities: Two main types Model 1 and Model 2, which are further classified into product type A or product type B. Now, consider how to represent the entity types. As was previously mentioned, this can be done in two ways: (i) by using the ENTITY module to define an entity type or (ii) by using user-defined attributes.

**TABLE 5.5 Activity 2.2 Data**

Model	Mean Rate	Type A(%), Process Time	Type B(%), Process Time
1	1 every 12 minutes	90%, expo(14) minutes	10%, expo(12) minutes
2	1 every 22 minutes	70%, expo(16) minutes	30%, expo(10) minutes

Either method can be used in this situation, but by using user-defined attributes, you can more easily facilitate some of the answers to some of the other modeling recipe questions.

Now, it is time to address the question: *What are the attributes of the entities?* This question is partly answered already by the decision to use user-defined attributes to distinguish between the types of entities. Let us define two attributes *myModel* and *myType* to represent the classification by model and by product type for the entities. For example, if *myModel* = 1, then the entity is of type model configuration 1. To continue addressing the required attributes, you need to consider the answers to the following questions:

- *What data do the entities need as they move through the model? Can they carry the data with them as they move or can the information be shared across entities?*
- *What data belongs to the system as a whole?*

The answer to the first question is that the parts need their processing times. The second question can have multiple answers, but for this model, the distributions associated with the processing times need to be known and this information does not change within the model. This should be a hint that it can be stored globally. In addition, the processing time needs to be determined based on different distributions at each of the two locations.

While you might first think of using variables to represent the processing time at each location, you will soon realize that there is a problem with this approach. Assume that a variable holds the processing time for the entity that is currently processing at location 1. While that entity is processing, another entity can come along and overwrite the variable. This is clearly a problem. The better approach is to realize that the entities can carry their processing times with them after the processing times have been computed. This should push you toward the use of an attribute to hold the processing time. Let us decide to have an attribute called *myProcessingTime* that can be used to hold the computed processing time for the entity before it begins processing. If data is not to be carried by the entities, but it still needs to be accessed, then this is a hint that it belongs to the system as a whole. In this case, the processing time distributions need to be accessed by any potential entity.

How can the entities be created? The problem description gives information about two Poisson arrival processes. While there are other ways to proceed, the most straightforward approach is to use a CREATE module for each arrival process. If this is done, one CREATE module will create the entities having model configuration 1 and the other CREATE module will create the entities having model configuration 2. How do you assign data to the entities? An ASSIGN module can be used for this task, but more importantly, the model configuration, the product type, and the processing time all will need to be assigned. In particular, the product type can be randomly assigned according to a given distribution based on the model configuration. In addition, the processing time is the sum of 10 draws from the processing time distribution. Thus, a way to compute the final processing time for the entity before it starts processing must be determined.

Exhibit 5.2 shows the basic pseudo-code for the example. After creation, the entities will have their model and type assigned. Then, they will proceed for processing. Prior to processing at the first station, the processing time is determined. In the pseudo-code, a WHILE-ENDWHILE construct is used to sum up the processing time. After the processing time has been determined, the entity delays for the processing time. When the processing is done at the first station, the processing time for the second station is used. Here the processing time is determined in an iterative manner by using a go to construct coupled

with an if statement. After the processing time has been determined, the entity again delays for the processing before being disposed.

---

**Exhibit 5.2 Iterative Looping Example**


---

```

CREATE 10 entities with TBA EXPO(12)
ASSIGN model 1 attributes
    myModel=1
    myType=DISC(.9,1,1,2)
END ASSIGN
GOTO label A

CREATE 10 entities with TBA EXPO(22)
ASSIGN model 2 attributes
    myModel=2
    myType=DISC(.7,1,1,2)
END ASSIGN
GOTO label A

A: determine the processing time for first station
ASSIGN initialize the loop
    vCounter=1
    vNumIterations=10
    myProcessingTime=0
END ASSIGN
WHILE vCounter ≤ vNumIterations
    ASSIGN
        myProcessingTime=myProcessingTime + ePTime(myModel, myType)
        vCounter=vCounter + 1
    END ASSIGN
END WHILE
DELAY for myProcessingTime
ASSIGN
    vCounter=0
    vNumIterations=10
    myProcessingTime=0
END ASSIGN

B: determine processing time for second station
ASSIGN
    myProcessingTime=myProcessingTime + ePTime(myModel, myType)
    vCounter=vCounter + 1
END ASSIGN
DECIDE if vCounter ≤ vNumIterations
    true: GOTO B
    false: GOTO C
END DECIDE

C:
DELAY for myProcessingTime
DISPOSE

```

---

The model will follow the outline presented within the pseudo-code. To support the modeling within the Arena™ environment, you must first define a number of variables and other data.

Variable - Basic Process							
	Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics
1	vCounter			Real	System	0 rows	<input type="checkbox"/>
2	vNumIterations			Real	System	1 rows	<input type="checkbox"/>
3	vMTCOUNT	2	2	Real	System	0 rows	<input type="checkbox"/>
4	vPTLocation1	2	2	Real	System	0 rows	<input type="checkbox"/>
5	vPTLocation2	2	2	Real	System	0 rows	<input type="checkbox"/>

Double-click here to add a new row.

Figure 5.32 Defining the variables.

**5.7.3.2 VARIABLE Module** Begin by defining the variables to be used in the model (Figure 5.32). Open the Arena™ environment and define following variables using the VARIABLE module.

- *vCounter* used to count the number of times the entity goes through the WHILE-ENDWHILE loop, a scalar variable. This WHILE-ENDWHILE loop will be used to compute the processing time at each location.
- *vNumIterations* used to indicate the total number of times to go through the WHILE-ENDWHILE loop, a scalar variable. This should be initialized to the value 10.
- *vMTCOUNT* used to assist with counting and displaying the number of entities of each model/type combination, a two-dimensional variable (two rows, two columns).
- *vPTLocation1* used to assist with displaying the processing time of each model/type combination, a two-dimensional variable (two rows, two columns).
- *vPTLocation2* used to assist with displaying the processing time of each model/type combination, a two-dimensional variable (two rows, two columns).

The example specifies that there is a different distribution for each model/product type combination. Thus, we need to be able to store these distributions. A distribution within Arena™ is considered a mathematical expression. A list of Arena™ distributions is found in Table C.1. In the table, the function name is given along with the parameters required by the function. The [,Stream] indicates an optional parameter that helps in controlling the randomness of the function. In addition to using distributions in expressions, you can use logical constructs and mathematical operators to build expressions. Table C.3 provides a list of mathematical functions available to the modeler. To represent the model/product type distributions, the EXPRESSION data module can be used.

**5.7.3.3 EXPRESSION Module** While you can build expressions using the previously mentioned functions and operators, a mechanism is needed to hold the expressions in memory, and in particular, for this problem, the appropriate distribution must be looked up based on the model type and the product type. The EXPRESSION module found on the Advanced Process template allows just this functionality. Expressions as defined in an EXPRESSION module are names or labels assigned to expressions. An expression defined in an EXPRESSION module is substituted by Arena™ wherever the named expression appears in the model. The EXPRESSION module also allows for the modeler to define an array of expressions.

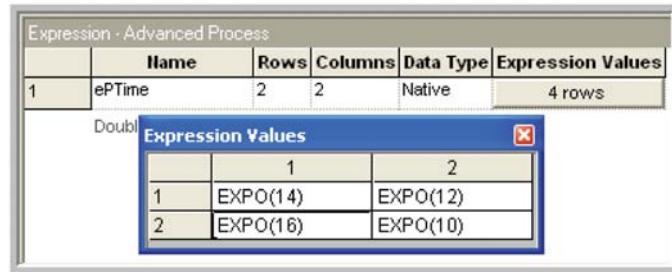


Figure 5.33 EXPRESSION module's spreadsheet view.

Create - Basic Process								
	Name	Entity Type	Type	Value	Units	Entities per Arrival	Max Arrivals	First Creation
1	Create Model Type 1	Entity 1	Random (Expo)	12	Minutes	1	10	0.0
2	Create Model Type 2	Entity 1	Random (Expo)	22	Minutes	1	10	0.0

Figure 5.34 CREATE module's spreadsheet view.

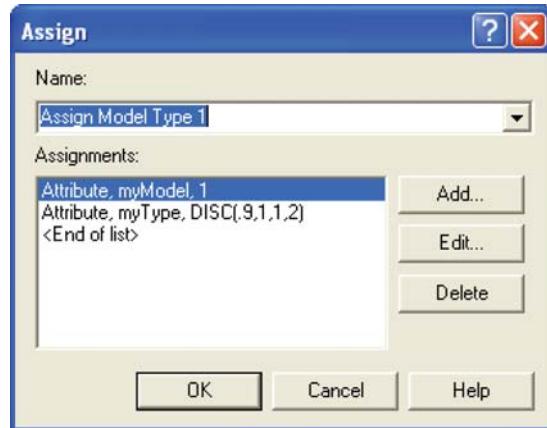
Go to the Advanced Process panel and define the expressions used in the model by clicking on the EXPRESSION module in the project bar, defining the name of the expression, *ePTime*, to have two rows and two columns, where the row designates the model type and the column designates the product type. Use the spreadsheet view to enter the exponential distribution with the appropriate mean values as indicated in Figure 5.33.

This creates an arrayed expression. This is similar to an arrayed variable, except that the value stored in the array element can be any valid Arena™ expression. Now, if you have the basic data predefined for the model, you can continue with the model building by starting with the CREATE modules. Since the model is large, you will begin by laying down the first part of the model. You should place the two CREATE modules and the two ASSIGN modules into the model window as shown in Figure 5.31. Fill in the create modules as indicated in Figure 5.34. This example model only requires 10 products of each model configuration to be created.

The ASSIGN modules can be used to define the attributes and assign their values. In the ASSIGN modules, the values of the *myModel* and *myType* attributes are assigned. There are two different models 1 and 2. Also, each model is assigned a type, dependent upon a specific probability distribution for that model. The DISC() function can be used to represent the model type. Open up the first ASSIGN module for model type 1 and make it look like the ASSIGN module show in Figure 5.35.

For variety, you can use the spreadsheet view to fill out the second ASSIGN module. Click on the ASSIGN module in the model window. The corresponding ASSIGN module will be selected in the spreadsheet view. Click on the assignment rows to get the assignments window and fill it in as indicated in Figure 5.36.

**5.7.3.4 Hierarchical Submodels** In this section, the middle portion of the model as indicated in Figure 5.37, which includes a submodel, will be built. A submodel in Arena™ is a named collection of modules that is used to organize the main model. Go to the Object menu and choose Submodel > Add Submodel. Place the cross-haired cursor that appears in the model window at the desired location and connect your ASSIGN modules to it as indicated in the figure. By right-clicking on the submodel and selecting the properties menu



**Figure 5.35** ASSIGN module's dialog box for model type 1.

**Assignments**

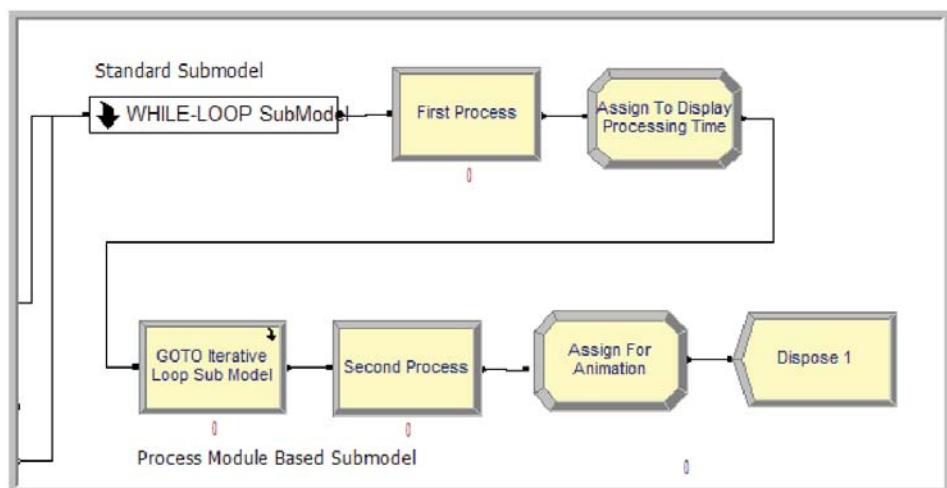
Type	Attribute Name	New Value
Attribute	myModel	2
Attribute	myType	DISC(.7,1,1,2)

Double-click here to add a new row.

**Assign - Basic Process**

Name	Assignments
Assign Model Type 1	2 rows
Assign Model Type 2	2 rows
Assign For Animation	2 rows
Assign To Display Processing Time	1 rows

**Figure 5.36** Second ASSIGN module in spreadsheet view.



**Figure 5.37** First location processing.

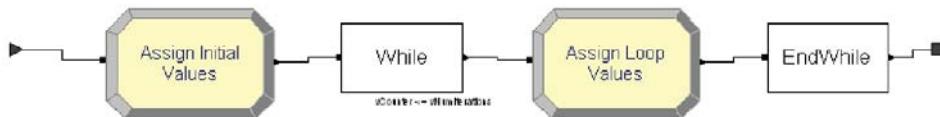


Figure 5.38 First location processing submodel.

Assignments				
	Type	Variable Name	Attribute Name	New Value
1	Variable	vCounter	Attribute 1	1
2	Variable	vNumIterations	Attribute 2	10
3	Attribute	Variable 3	myProcessingTi	0

Figure 5.39 ASSIGN within first submodel.

item, you can give the submodel a name. This submodel will demonstrate the use of a WHILE-ENDWHILE loop in Arena™. The WHILE-ENDWHILE construct will be used as an iterative looping technique to compute the total processing time according to the number of iterations (10) for the given model/type combination.

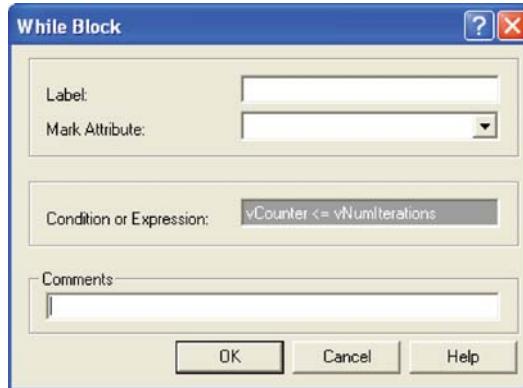
**5.7.3.5 WHILE-ENDWHILE Blocks** Open up the submodel by double-clicking on it and add the modules (ASSIGN, WHILE, ASSIGN, and ENDWHILE) to the submodel as indicated in Figure 5.38. You will find the WHILE and ENDWHILE blocks on the BLOCKS panel. If you have not already done, you should attach the BLOCKS panel by going to the Basic Process panel and right-clicking, choose Attach, and then select the file called *Blocks.tpo*. A new panel of blocks will appear for use in Arena™.

Initialize the counter in the first Assign block:

1. The counter can be an attribute or a variable
2. If the counter is an attribute, then every entity has this attribute and the value will persist with the entity as it moves through the model. It may be preferable to have the counter as an attribute for the reason given in item (3) below.
3. If the counter is a (global) variable, then it is accessible from anywhere in the model. If the WHILE-ENDWHILE loop contains a block that can cause the entity to stop moving (e.g., DELAY and HOLD), then it may be possible for another entity to change the value of the counter when the entity in the loop is stopped. This may cause unintended side effects, for example, resetting the counter would cause an infinite loop. Since there are no time delays within this WHILE-ENDWHILE loop example, there is no danger of this. Remember that there can only be one entity moving (being processed by the simulation engine) at anytime while the simulation is running.

Open up the first ASSIGN module and make it look like that shown in Figure 5.39. Notice how the variables are initialized prior to the WHILE. In addition, the *myProcessingTime* attribute has been defined and initialized to zero.

Now the counter must be checked in the condition expression of the WHILE block. The WHILE and ENDWHILE blocks can be found on the Blocks panel. Open up the WHILE



**Figure 5.40** WHILE block within first submodel.

Assignments				
	Type	Variable Name	Attribute Name	New Value
1	Attribute	Variable 1	myProcessingTime	myProcessingTime + ePTime(myModel, myType)
2	Variable	vCounter	Attribute 2	vCounter + 1
Double-click here to add a new row.				

**Figure 5.41** Second ASSIGN module within first submodel.

block and make it look like that shown in Figure 5.40. When the variable *vCounter* is greater than *vNumIterations*, the WHILE condition will evaluate to false and the entity will exit the WHILE-ENDWHILE loop.

Make sure to update the value of the counter variable or the evaluation expression within the body of the WHILE and ENDWHILE loop; otherwise you may end up with an infinite loop. Open up the second ASSIGN module and make the assignments as shown in Figure 5.41. In the first assignment, the attributes *myModel* and *myType* are used to index into the *ePTime* array of expressions. Each time the entity hits this assignment statement, the value of the *myProcessingTime* attribute is updated based on the previous value, resulting in a summation. The variable *vCounter* is incremented each time to ensure that the WHILE loop will eventually end.

There is nothing to fill in for the ENDWHILE block. Just make sure that it is connected to the last ASSIGN module and to the exit point associated with the submodel. Right-clicking within the submodel will give you a context menu for closing the submodel. After the entity has exited the submodel, the appropriate amount of processing time has been computed within the attribute *myProcessingTime*. You now need to implement the delay associated with processing the entity for the computed processing time.

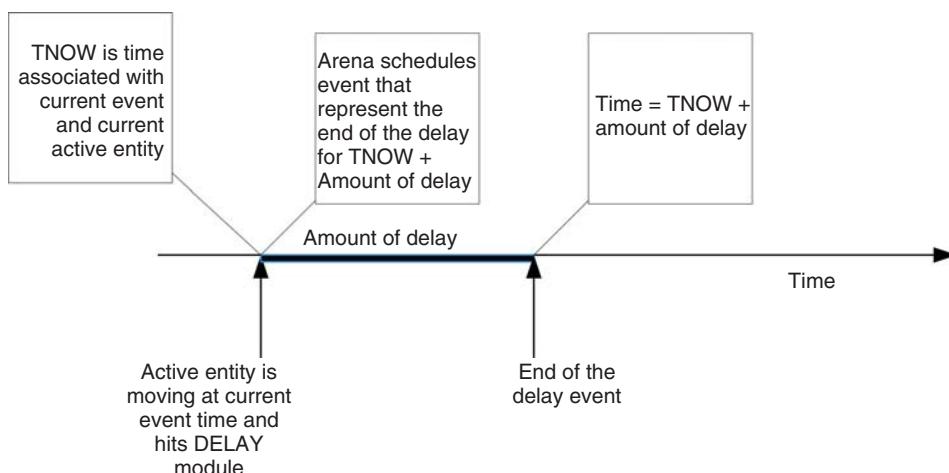
**5.7.3.6 PROCESS Module Delay Option** Within Arena™, only the active entity is moving through the modules at any time. The active entity is the entity that the simulation executive has determined is associated with the current event. At the current event time, the active entity executes the modules along its flow path until it is delayed or cannot proceed due to a condition in the model. When the active entity is time delayed, an event is placed on the event calendar that represents the time that the delay will end. In other words, a future

event is scheduled to occur at the end of the delay. The active entity is placed in the future events list and yields control to the next entity that is scheduled to move within the model. The simulation executive then picks the next event and the associated entity and tells that entity to continue executing modules. Thus, only one entity is executing model statements at the current time and module execution only occurs at event times. Arena<sup>TM</sup> moves from event time to event time, executing the associated modules. This concept is illustrated in Figure 5.42 where a simple delay is scheduled to occur in the future. It is important to remember that while the entity associated with the delay is waiting for its delay to end, other events may occur, causing other entities to move.

There are two basic ways to cause a delay for an entity, the PROCESS module and the DELAY module. The PROCESS module is found on the Basic process template. The DELAY module is found on the Advanced Process template panel. This example illustrates the use of the PROCESS module. The next step in building the model is to place the PROCESS module after the submodel as shown in Figure 5.37. Open up the PROCESS model and select the “Delay” option for the *Action* text drop down box in the logic area of the dialog box. Now you can indicate the length of the delay. Arena<sup>TM</sup> allows the user to select the type of delay. In this case, you want to delay by the amount of time computed for the processing time. This value is stored in the *myProcessingTime* attribute. Within the *Delay Type* area, choose expression and in the expression text field, fill in *myProcessingTime* as shown in Figure 5.43. Be careful to appropriately set the time units (minutes) associated with the delay. A common mistake is to not set the units and have too long or too short a delay. The too long a delay is often caught during debugging because the entities take such a long time to leave the module.

Now you will add an ASSIGN module to facilitate the displaying of the values of variables in the model window during the simulation. This will also illustrate how arrays can be used within a model. Using the spreadsheet view, make the ASSIGN module look like that shown in Figure 5.44.

The values of *attributes* cannot be displayed using the variable animation constructs within Arena<sup>TM</sup>. So, in this ASSIGN module, an array variable is used to temporarily assign



**Figure 5.42** Scheduling a delay.

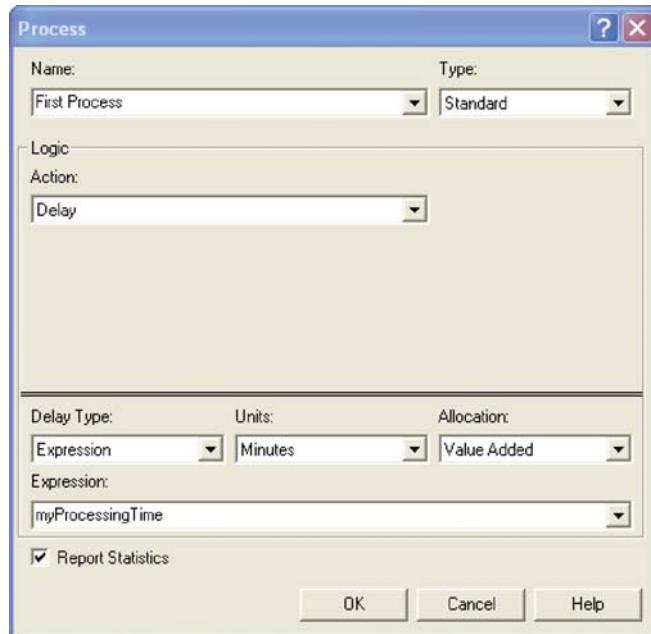


Figure 5.43 The PROCESS module.

	Type	Variable Name	Row	Column	New Value
1	Variable Array (2D)	vPTLocation1	myModel	myType	myProcessingTime

Double-click here to add a new row.

Figure 5.44 ASSIGN module with arrays.

the value of the attribute to the proper location of the array variable so that it can be displayed. Go to the toolbar area in Arena™ and right-click. Make sure that the Animate toolbar is available. From the Animate toolbar, choose the button that has a 0.0 indicated on it. You will get a dialog for entering the variable to display. Fill out the dialog box so that it looks like Figure 5.45. After filling out the dialog and choosing OK, you will need to place the cross-hair somewhere near the ASSIGN module in the model window. In fact, the element can be placed anywhere in the model window. Repeat this process for the other elements of the vPTLocation1 array.

Now, the model can be completed. Lay down the modules indicated in Figure 5.37 to complete the model. In Figure 5.37, the second submodel uses a PROCESS-based submodel. It looks like a PROCESS module and is labeled “GOTO Iterative Loop Sub Model,” see Figure 5.46. To create this submodel, lay down a PROCESS module and change its type to submodel. In what follows, you will examine a different method for looping within the second submodel. Other than that, the rest of the model is similar to what has already been completed. The second submodel has the modules given in Figure 5.47. Place and connect the ASSIGN, DECIDE, and ASSIGN modules as indicated in Figures 5.48 and 5.49. In



Figure 5.45 Animate variable dialog box.

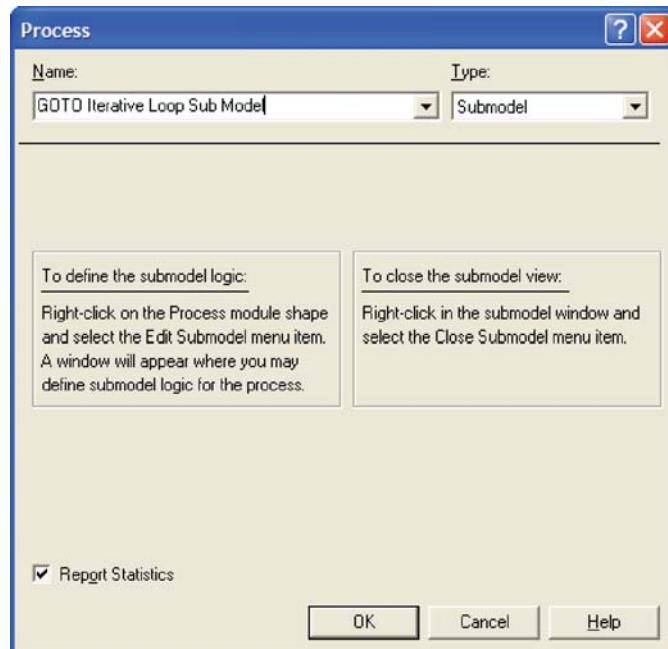


Figure 5.46 PROCESS-based submodel.

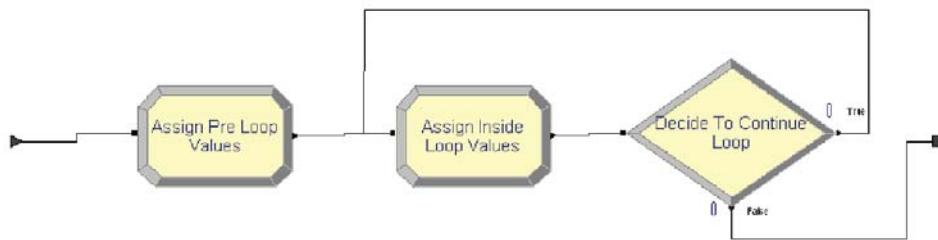


Figure 5.47 Second submodel.

Assignments				
	Type	Variable Name	Attribute Name	New Value
1	Variable	vCounter	Attribute 1	0
2	Variable	vNumIterations	Attribute 2	10
3	Attribute	Variable 3	myProcessingTime	0

(a)

Assignments				
	Type	Variable Name	Attribute Name	New Value
1	Attribute	Variable 1	myProcessingTime	myProcessingTime + ePTime(myModel, myType)
2	Variable	vCounter	Attribute 2	vCounter + 1

(b)

Figure 5.48 ASSIGN modules in second submodel. (a) Assign preloop values. (b) Assign inside loop values.

Decide				
Name:	Type:			
Decide To Continue Loop	2-way by Condition			
If:	Named:	Is:		
Variable	vCounter	$\leq$		
Value:	vNumIterations			
<input type="button" value="OK"/> <input type="button" value="Cancel"/> <input type="button" value="Help"/>				

Figure 5.49 DECIDE module in second submodel.

	Type	Variable Name	Row	Column	New Value
1	Variable Array (2D)	vPTLocation2	myModel	myType	myProcessingTime
2	Variable Array (2D)	vMTCount	myModel	myType	vMTCount(myModel,myType) + 1

Figure 5.50 Last ASSIGN module—assign for animation.

this logic, the entity initializes a counter for counting the number of times through the loop and updates the counter. An If/Then DECIDE module is used to redirect the entity back through the loop the appropriate number of times. This is GOTO programming!

Again, be sure to use logic that will change the tested condition (in this case *vCounter*) so that you do not get an infinite loop. I prefer the use of the WHILE-ENDWHILE blocks instead of this “go to” oriented implementation, but it is ultimately a matter of taste.

In general, you should be careful when implementing logic in which the entity simply loops around to check a condition. As mentioned, you must ensure that the condition can change to prevent an infinite loop. A common error is to have the entity check a condition that can be changed from somewhere else in the model, for example, a queue becomes full. Suppose you have the entity looping around to check if the queue is not full so that when the queue becomes full, the entity can perform other logic. This seems harmless enough; however, if the looping entity does not change the status of the queue, then it will be in an infinite loop! Why? While you might have many entities in the model at any given simulated time, only *one* entity can be moving at any time. You might think that one of the other entities in the model may enter the queue and thus change its status. If the looping entity does not enter a module that causes it to “hand off the ball” to another entity, then no other entity will be able to move to change the queue condition. If this occurs, you will have an infinite loop.

If the looping entity does not explicitly change the checked condition, then you must ensure that the looping entity passes control to another entity. How can this be achieved? The looping entity must enter a blocking<sup>3</sup> module that allows another entity to get to the beginning of the future events list and become the active entity. There are a variety of modules which will cause this, for example, HOLD, SEIZE, and DELAY. Another thing to remember, even if the looping entity enters a blocking module, your other simulation logic must ensure that it is at least possible to have the condition change. If you have a situation where an entity needs to react to a particular condition changing in the model, then you should consider investigating the HOLD and SIGNAL modules, which were designed specifically for these types of situations.

The second PROCESS module is exactly the same as the first PROCESS module. In fact, you can just copy the first PROCESS module and paste the copy into the model at the appropriate location. If you do copy and paste a module in this manner, you should make sure to change the name of the module because module names must be unique within Arena™. The last ASSIGN module is just like the previously explained example. Complete the ASSIGN module as shown in Figure 5.50. Complete the animation of the model in a similar manner as you did to show the processing time at the first location, except in this case show the counts captured by the variable *vMTCount*.

<sup>3</sup>A blocking module is a module that causes the forward movement of an entity to stop in some manner.

To run the model, execute the model using the “VCR” run button. It is useful to use the step command to track the entities as they progress through model. You will be able to see that the values of the variables change as the entities go through the modules.

To recap, in this example, you learned about attributes and how they are attached to entities. In addition, you saw how to share global information through the use of arrays of variables and arrays of expressions. The concept of scheduling events via the use of the delay option within the PROCESS panel was also introduced. Finally, you also refreshed your memory on basic programming constructs such as decision logic and iterative processing.

## 5.8 BATCHING AND SEPARATING ENTITIES

Of the flowchart modules on the Basic Process panel, there are two remaining modules to discuss: BATCH and SEPARATE. The BATCH module allows entities to be grouped together into a temporary or permanent representative entity. A representative entity is an entity that consists of a group of entities that can travel together and be processed as if there was only one entity. A temporary representative entity can be split apart, by a SEPARATE module, so that the individual entities can again be processed. For a permanent representative entity, the batched individual entities cannot be separated. You can use a permanent entity when you no longer need the information associated with the individual entities within the group. You can think of a temporary entity as having an open bag that holds the batched members in the group, and a permanent entity as having a closed bag. Temporary entities are useful for modeling a manufacturing system when items are assembled and disassembled during processing. A permanent entity is useful in modeling a situation where the assembly is complete.

The batching of entities can be performed based on a number of criteria. For example, the batch might form when at least 10 entities are available. Thus, an important consideration in batching is how to determine the size of the batch. In addition, batches might be formed based on entities that have similar attributes. For example, suppose red and blue paper folders are being manufactured in batches of five of a specific color for sale by color. The batching of entities is also useful in synchronizing the movement of entities. As indicated above, when potential members of a batch enter the BATCH module, they wait until the batching conditions have occurred before they continue their movement with the representative entity in the model. Because of this, the BATCH module has a queue that holds the entities that are waiting to be batched. In the case of a temporary representative entity, the SEPARATE module provides the functionality to split apart the representative entity into the individual entities. The SEPARATE module can also *duplicate* or *clone* the incoming entity. This provides an additional mechanism to introduce new entities into the model, that is, another way of creating entities. In the following examples, the BATCH and SEPARATE modules will be examined to investigate some of the rules involved in the use of these modules.

### 5.8.1 Example: Tie-Dye T-Shirts

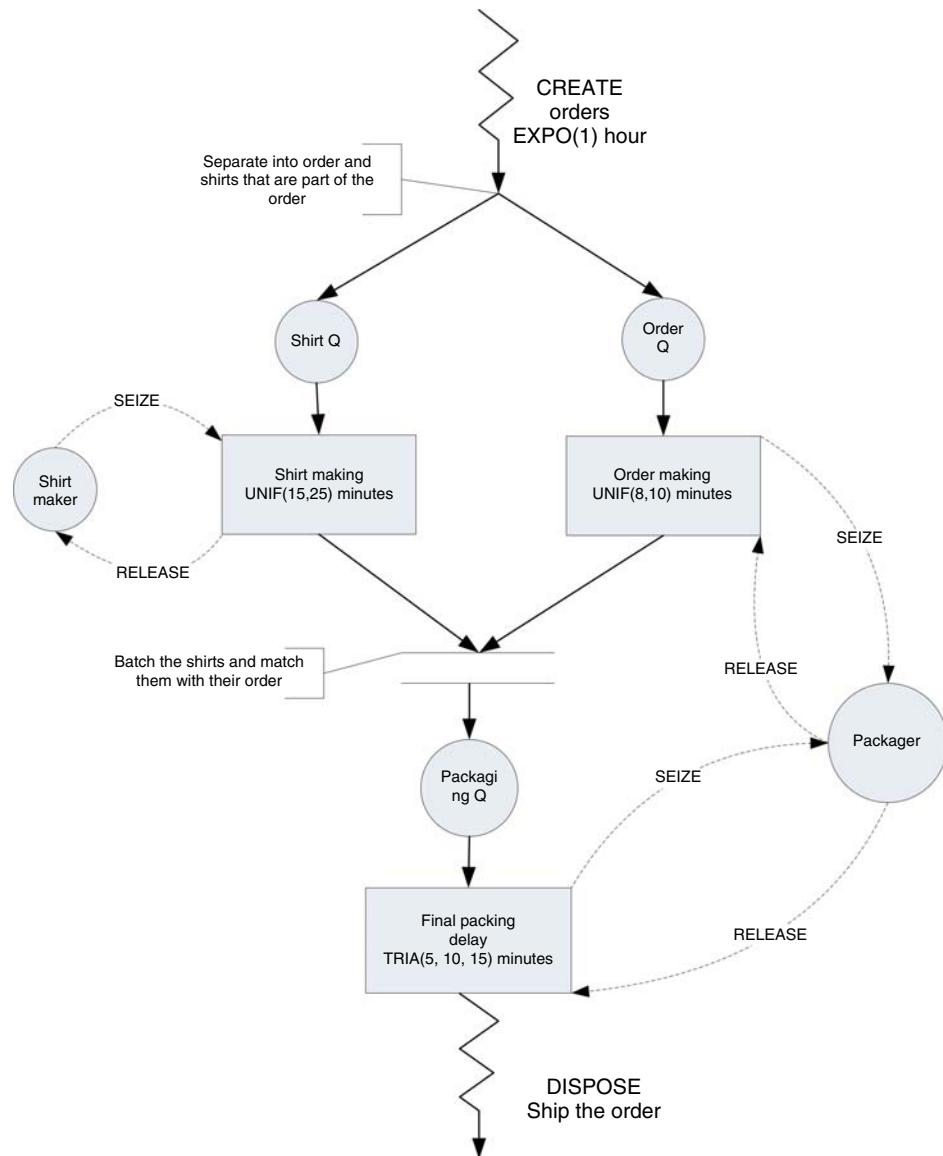
Suppose production orders for tie-dye T-shirts arrive to a production facility according to a Poisson process with a mean rate of 1 per hour. There are two basic psychedelic designs involving either red or blue dye. For some reason, the blue shirts are a little more popular than the red shirts so that when an order arrives, about 70% of the time it is for the blue

dye designs. In addition, there are two different package sizes for the shirts, 3 and 5 units. There is a 25% chance that the order will be for a package size of 5 and a 75% chance that the order will be for a package size of 3. Each of the shirts must be individually handmade to the customer's order design specifications. The time to produce a shirt (of either color) is uniformly distributed within the range of 15–25 minutes. There are currently two workers who are setup to make either shirt. When an order arrives to the facility, its type (red or blue) is determined and the pack size is determined. Then, the appropriate number of white (un-dyed) shirts is sent to the shirt makers with a note pinned to the shirt indicating the customer order, its basic design, and the pack size for the order. Meanwhile, the paperwork for the order is processed and a customized packaging letter and box is prepared to hold the order. It takes another worker between 8 and 10 minutes to make the box and print a custom thank you note. After the packaging is made, it waits prior to final inspection for the shirts associated with the order. After the shirts are combined with the packaging, they are inspected by the packaging worker which is distributed according to a triangular distribution with a minimum of 5 minutes, a most likely value of 10 minutes, and a maximum value of 15 minutes. Finally, the boxed customer order is sent to shipping.

**5.8.1.1 Conceptualizing the Model** Before proceeding, you might want to jot down your answers to the modeling recipe questions and then you can compare how you are doing with respect to what is presented in this section. The following are the modeling recipe questions:

- What is the system? What information is known by the system?
- What are the required performance measures?
- What are the entities? What information must be recorded or remembered for each entity? How are entities introduced into the system?
- What are the resources that are used by the entities? Which entities use which resources and how?
- What are the process flows? Sketch the process or make an activity flow diagram.
- Develop pseudo-code for the situation.
- Implement the model in Arena™.

The entities can be conceptualized as the arriving orders. Since the shirts are processed individually, they should also be considered as entities. In addition, the type of order (red or blue) and the size of the order (3 or 5) must be tracked. Since the type of the order and the size of the order are properties of the order, attributes can be used to model this information. The resources are the two shirt makers and the packager. The flow is described in the scenario statement: orders arrive, shirts made, meanwhile packaging is made. Then, orders are assembled, inspected, and finally shipped. It should be clear that a CREATE module setup to generate Poisson arrivals can create the orders, but if shirts are entities, how should they be created? To do this, a SEPARATE module can be used to make the number of shirts required based on the size of the order. After this, there will be two types of entities in the model, the orders and the shirts. The shirts can be made and meanwhile the order can be processed. When the shirts for an order are made, they need to be combined together and then matched for the order. This implies that a method is required to uniquely identify the order. This is another piece of information that both the order and the shirt require. Thus, an attribute will be used to note the order number.



**Figure 5.51** Activity diagram for tie dye T-shirts example.

The activity diagram for this situation is given in Figure 5.51. After the order is created, the process separates into the order making process and the shirt making process. Notice that the orders and shirts must be synchronized together after each of these processes. In addition, the order making process and the final packaging process share the packager as a resource.

Exhibit 5.3 presents the representation for the activity diagram in pseudo-code. In the exhibit, a variable is incremented each time an order is created and then the value of the variable is assigned to the attribute representing the order. In addition, both the order type and the order size are assigned based on the probability distribution information that was given

in the problem. The parallel processing is represented by the two pseudo-code segments labeled A and B.

---

**Exhibit 5.3 Tie Dye T-Shirts Example**


---

```

CREATE orders every hour exponentially
ASSIGN vOrderNumber=vOrderNumber + 1
    myOrderNumber=vOrderNumber
    myOrderType=DISC(0.7, 1, 1.0, 2)
    myOrderSize=DISC(0.75, 3,1.0, 5)
END ASSIGN

SEPARATE into order and shirts based on myOrderSize
    Shirts GOTO Label A shirt making
    Order GOTO Label B order making
END SEPARATE

A:
PROCESS
    SEIZE packager
    DELAY UNIF(8,10) minutes
    RELEASE packager
END PROCESS
Wait until shirts for matching order are completedWhen
match occurs, GOTO Label C, order forming

B:
PROCESS
    SEIZE shirt maker
    DELAY UNIF(15,20) minutes
    RELEASE shirt maker
END PROCESS
BATCH shirts on order togetherWait until order for the
matching shirts are completedWhen match occurs, GOTO
Label C, order forming

C:
BATCH order and shirts together
PROCESS
    SEIZE packager
    DELAY TRIA(5,10,15) minutes
    RELEASE packager
END PROCESS
DISPOSE: Ship order out of the system

```

---

**5.8.1.2 Building the Model** The completed Arena<sup>TM</sup> model is given in Figure 5.52. Notice how there is a parallel structure in the middle of the model. This is where the orders are processed separately from the shirts. This is quite a common pattern in simulation modeling.

Take the time now to drag and drop the necessary modules to your model area. Each of the modules will be discussed in what follows. The MATCH module has not been mentioned previously. It is found on the Advanced Process panel. The MATCH module will hold entities until enough entities to meet the match requirement enter the queues. This

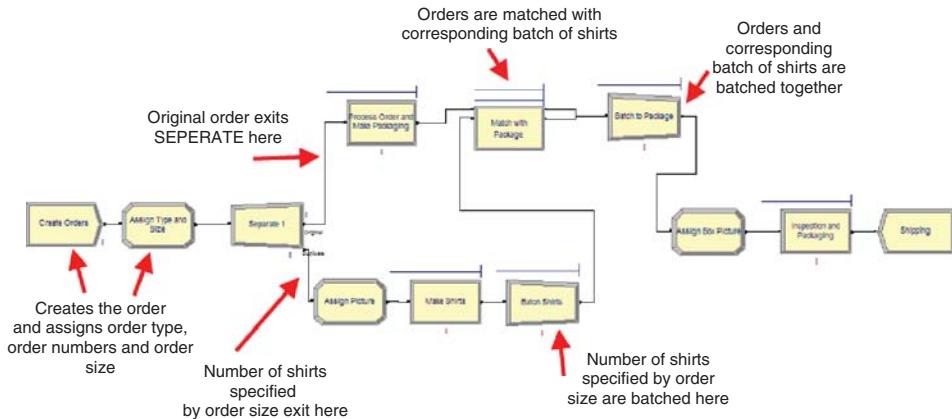


Figure 5.52 Overall tie-dye model

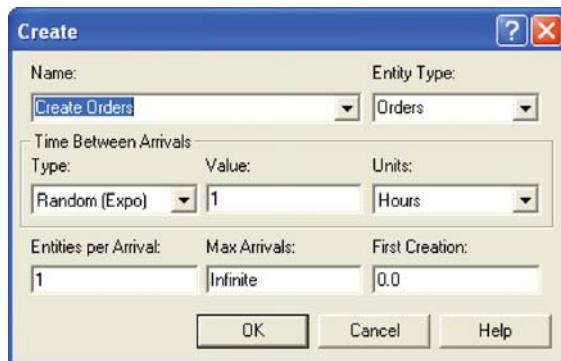


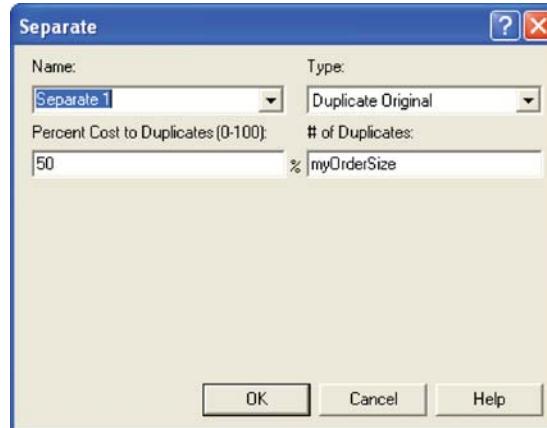
Figure 5.53 CREATE model for tie-dye shirt model.

Assignments				
	Type	Variable Name	Attribute Name	
1	Variable	vOrderNumber	Attribute 1	vOrderNumber + 1
2	Attribute	Variable 2	myOrderNumber	vOrderNumber
3	Attribute	Variable 3	myOrderType	DISC(0.7, 1, 1.0, 2)
4	Attribute	Variable 4	myOrderSize	DISC(0.75, 3, 1.0, 5)

Figure 5.54 Assigning the order number, type, and size.

module will be used to match the processed orders with the already batched shirts for the orders. The CREATE module is essentially the same as you have already seen. Open up your CREATE module and fill it out as shown in Figure 5.53.

In the ASSIGN module of Figure 5.54, a variable is used to count each order as it arrives. This unique number is then assigned to the *myOrderNumber* attribute. This attribute will be used to uniquely identify the order within the model. Then, the DISC() random distribution function is used to assign the type of order (Blue = 1, Red = 2) according to the given



**Figure 5.55** Using the SEPARATE module to create the shirts.

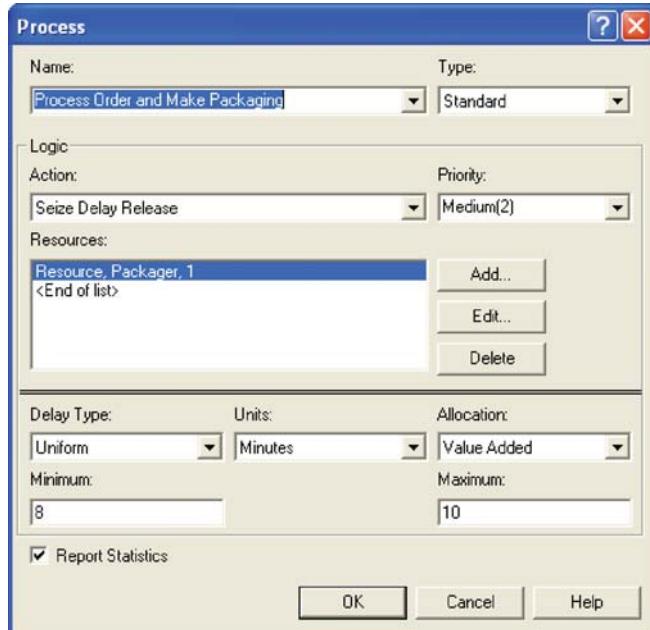
Resource - Basic Process						
	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use
1	ShirtMakers	Fixed Capacity	2	0.0	0.0	0.0
2	Packager	Fixed Capacity	1	0.0	0.0	0.0
Double-click here to add a new row.						

**Figure 5.56** Defining the resources.

probabilities. Notice that the concept of blue and red are mapped to the numbers 1 and 2, respectively. This is often the case within Arena™, since attributes and variables can only be real numbers. The DISC() distribution is used to randomly assign the size of the order. This will be remembered by the entity within the *myOrderSize* attribute.

After you have filled in the dialog boxes as shown, you can proceed to the SEPARATE module. The SEPARATE module has two options: *Split Existing Batch* and *Duplicate Original*. The *Duplicate Original* option is used here. The entity that enters the SEPARATE module is an order; however, after proceeding through the module, both orders and shirts will depart from the module. Open up the SEPARATE module and notice the text box labeled *# of Duplicates* as shown in Figure 5.55. This field indicates how many entities will be created by the SEPARATE module. It can be any valid Arena™ expression. The size of the order should be used to indicate how many entities to create. For example, if the *myOrderSize* attribute was set to 3, then three additional entities will be cloned or duplicated from the original entering entity. For modeling purposes, these entities will be conceptualized as the shirts. Do not worry about the text field in the dialog box relating to cost attributes at this time.

When an entity enters the SEPARATE module, the original will exit along the original exit point and the duplicate (or split off) entities will exit along the duplicate exit point. Since the created entities are duplicates, they will have the same values for all attributes as the original. Thus, each shirt will know its order type (*myOrderType*) and its order size (*myOrderSize*). In addition, each shirt will also know which order it belongs to through the



**Figure 5.57** Seizing, delaying, and releasing the packager.

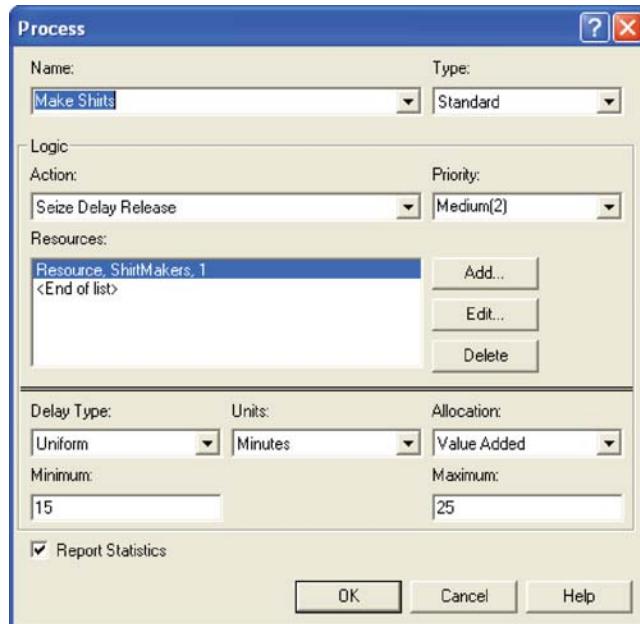
(*myOrderNumber*) attribute. These shirt attributes will be used when combining the orders with their shirts after all the processing is complete.

Next, you should define and use the resources in the model. First, you should add the resources to the model using the resource datasheet view as shown in Figure 5.56. Notice here that there are two units of capacity for the shirt maker resource to represent the two workers involved in this process.

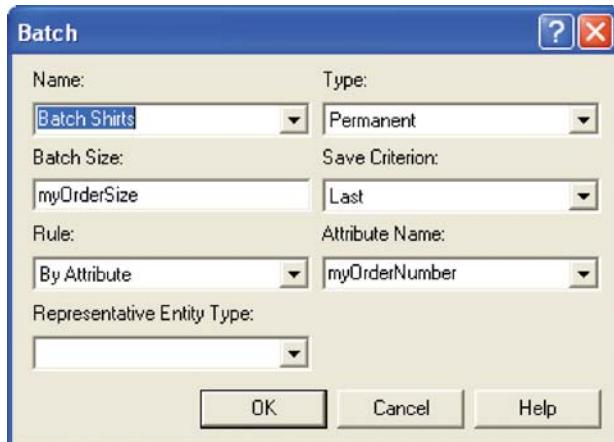
The packager is used to process the orders along the original entity path. Figure 5.57 shows the dialog box for using the packager. This is similar to how the pharmacist was implemented in the drive through pharmacy model. Fill out both PROCESS modules as shown in Figures 5.57 and 5.58.

After the order's packaging is complete, it must wait until the shirts associated with the order are made. As seen in Figure 5.52, the orders go to a MATCH module, where they will be matched by the attribute *myOrderNumber* with the associated group of shirts that exit the BATCH module named *Batch Shirts*. Open up the BATCH module that you previously placed and fill it out as shown in Figure 5.59. This BATCH module creates a permanent entity because the shirts do not need to be processed individually after being combined into the size of the order. The entities that enter this module are shirts. They can be red or blue. The shirts enter the module and wait until there are *myOrderSize* other entities also in the batch queue that have the same indicated attribute. Then these entities are combined together and leave as a permanent entity.

The attributes of the representative entity are determined by the selection indicated in the *Save Criterion* text box field. All the user-defined attributes of the representative entity are assigned based on the Save Criterion.



**Figure 5.58** Seizing, delaying, and releasing the shirt makers.

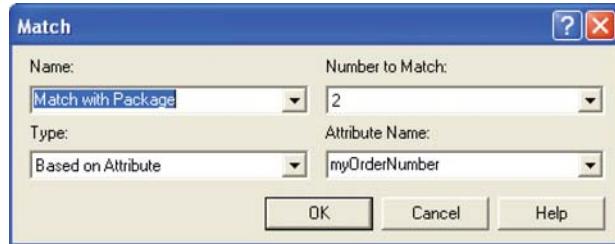


**Figure 5.59** Batching the shirts together.

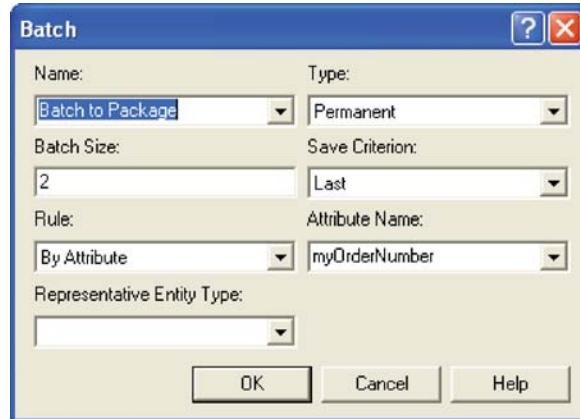
**First or Last** assigns the user-defined attributes based on the first/last entity forming the batch.

**Product** multiplies the value of each user-defined attribute among all entities in the batch and assigns the product to the corresponding attribute of the representative entity.

**Sum** performs the same action, adding instead of multiplying.



**Figure 5.60** Matching the order with the shirt group.

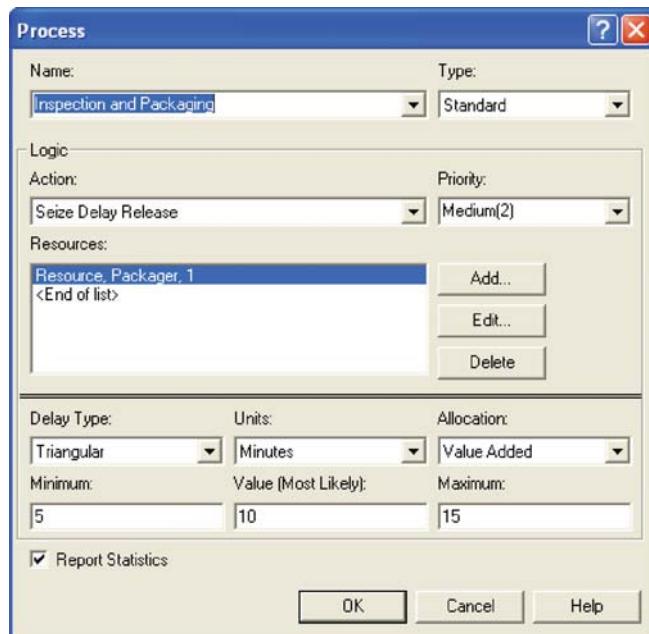


**Figure 5.61** Batching the matched order with the shirt group.

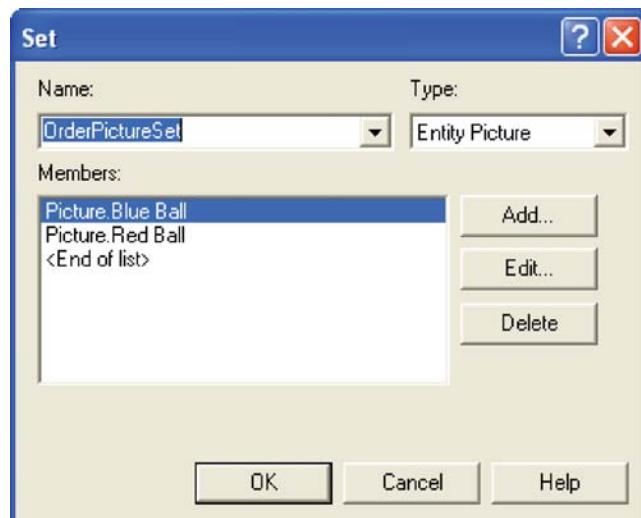
For a detailed discussion of how Arena™ handles the assigning of its special purpose attributes, you should refer to the help files on the BATCH module. In fact, reading the help files for this important module is extremely useful. In this example, *First* or *Last* can be used. When an entity leaves the BATCH module, it will be a group of *myOrderSize* shirts for specified order number. The group of shirts will exit the BATCH and then enter the MATCH module to be matched with the packaging for the order. The MATCH module in Figure 5.60 has two match queues. The order/packaging enters the top queue and groups of shirts enter the lower queue. Whenever an entity enters the module, Arena™ will check if the appropriate number of entities to match are available within each queue that meet the specified matching criteria. In this example,<sup>4</sup> the order number is used as the matching attribute so that the order/packaging will wait for the group of shirts (or vice versa). After they are matched, the matching entities are released from the MATCH module.

The group of shirts and the order/packaging are now synchronized. It is very common to use a BATCH module, as done here, to make the synchronization permanent by batching the entities together as per Figure 5.61. Since the order number is unique and the batch size is 2, the result of this BATCH module will be to combine the group of shirts and the order/packaging together into a permanent entity. After the order is completed, it is sent to a PROCESS module (Figure 5.62) where it is inspected and then sent to a DISPOSE module, which represents shipping.

<sup>4</sup>The MATCH module is actually not needed in this model and is shown only for illustrative purposes. Do you understand why?



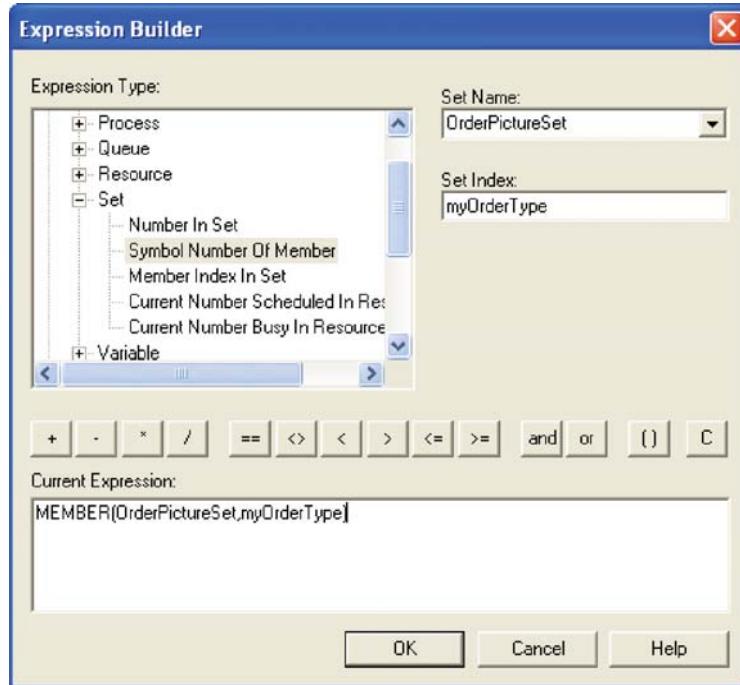
**Figure 5.62** Using the packager to do inspection.



**Figure 5.63** Defining a picture set.

The only items that have not been discussed are the two ASSIGN modules labeled *Assign 2* and *Assign 3*. In these modules, the animation picture of the entities is changed so that the operation of the SEPARATE, MATCH, and BATCH modules will be more visible in the animation. To do this, a SET is used to hold the pictures for each type of entity.

The SET module is found on the Basic Process panel as a data module. A set is a group of related (similar) objects that are held in a list within Arena™. In this instance, a set is



**Figure 5.64** Using the expression builder to index into a set.

Assignments			
	Type	Other	New Value
1	Other	Entity.Picture	MEMBER(OrderPictureSet,myOrderType)
Double-click here to add a new row			

**Figure 5.65** Assign picture: assigning an animation picture based on a set index.

defined to hold a list of animation pictures. Click on the SET module and use the edit via dialog option to add the picture of the blue ball and the red ball to a set with its type set to Entity Picture as shown in Figure 5.63. The order of the entries in the set matters. The first location in the set is associated with the blue ball and the second location in the set is associated with the red ball. This is done because the numbers 1 and 2 have been mapped to the type of order, Blue = 1 and Red = 2. The order type attribute can be used to index into the set to assign the appropriate picture to the entity.

In Figure 5.65, an assignment of type “Other” has been made to the Entity.Picture a special purpose attribute. The expression builder was used to build the appropriate expression for indexing into the set as shown in Figure 5.64. The resulting ASSIGN module is shown in Figure 5.66. Figure 5.66 indicates the ASSIGN module for directly assigning a picture

Assignments		
	Type	Entity Picture
1	Entity	Picture Box
Double-click here to add a new row.		

**Figure 5.66** Assign box picture: directly assigning an animation picture.

to an entity. In this case, since the shirts and packaging have been combined, the combined entity is shown as a box.

You should run the model for 8 hours and set the base time unit to minutes. When you run the model, use the animation slider bar to slow down the speed of the animation. You will see in the animation how the SEPARATE, MATCH, and BATCH modules operate. For this short period of time, it took about 75 minutes on an average to produce an order. You are asked to further explore the operation of this system in the exercises.

In this example, you learned that parallel processing can be easily accomplished within Arena™ by using a SEPARATE module. This facilitates the creation of other entity (types) with the model and allows them to proceed in parallel. The BATCH module shows how you can combine (and synchronize) the movement of a group of entities. In addition, the MATCH module also facilitates the synchronization of entity movement. Finally, you saw that Arena™ also has another data structure for holding information (besides variables and attributes) called a set. A set acts as a list of similar items and can be indexed to return the items in the set. The use of the set allowed an index to be used in an animation picture set to assign different pictures to the entities as they move through the model. This is especially useful in debugging and interpreting the actions of modules within Arena™.

## 5.9 SUMMARY

In this chapter, you have learned a great deal about how Arena™ can function as a programming language. While Arena™ provides an environment to build simulation models, this model building process requires the user to have a basic knowledge of programming to be productive. This chapter has made some analogies between common programming languages and Arena™ by showing you how typical programming aspects (defining variables, attributes, I/O, iteration, etc.) can be performed within Arena™.

In addition, all of the modules found in Arena's Basic Process panel, except for the SCHEDULE module have been discussed. The modules covered include the following:

- CREATE** Used to create and introduce entities into the model according to a pattern.
- DISPOSE** Used to dispose of entities once they have completed their activities within the model.
- PROCESS** Used to allow an entity to experience an activity with the possible use of a resource.
- ASSIGN** Used to make assignments to variables and attributes within the model.
- RECORD** Used to capture and tabulate statistics within the model.
- BATCH** Used to combine entities into a permanent or temporary representative entity.

**SEPARATE** Used to create duplicates of an existing entity or to split a batched group of entities.

**DECIDE** Used to provide alternative flow paths for an entity based on probabilistic or condition-based branching.

**VARIABLE** Used to define variables to be used within the model.

**RESOURCE** Used to define a quantity of units of a resource that can be seized and released by entities.

**QUEUE** Used to define a waiting line for entities whose flow is currently stopped within the model.

**ENTITY** Used to define different entity types for use within the model.

**SET** Used to define a list of elements within Arena<sup>TM</sup> that can be indexed by the location in the list.

In addition to Arena's Basic Process panel, the following constructs have been introduced.

**READWRITE** From the Advanced Process panel, this module allows input and output to occur within the model.

**FILE** From the Advanced Process panel, this module defines the characteristics of the operating system file used within a READWRITE module.

**EXPRESSION** From the Advanced Process panel, this module allows the user to define named logical/mathematical expressions that can be used throughout the model.

**DELAY** From the Advanced Process panel, this module allows an entity to experience a delay in movement via the scheduling of an event.

**MATCH** From the Advanced Process panel, this module allows entities to wait in queues until a user-specified matching criteria occurs.

**IF-ELSEIF-ELSE-ENDIF** From the Blocks panel, these modules allow standard logic-based flow of control.

**WHILE-ENDWHILE** From the Blocks panel, these modules allow for iterative looping.

**BRANCH** From the Blocks panel, this module allows probabilistic and condition-based path determination along with cloning of entities.

You also learned that Arena<sup>TM</sup> has a number of variables (e.g., TNOW), attributes (e.g., Entity.Type), and mathematical functions (e.g., NORM() and ABS ()) that can be used within a model.

To develop a program in Arena<sup>TM</sup>, you used data modules to define the elements to be used in the model and flowchart modules to specify the logical flow of the model. The flowchart style that Arena<sup>TM</sup> facilitates has its advantages and disadvantages. The primary advantage is that you can quickly build useful models within the environment without really knowing how to program. This is a great boon to the use of simulation technology. The primary disadvantage is that the flowchart paradigm makes it difficult to organize and develop code that is well structured.

Because of this disadvantage, I strongly encourage you to plan your simulation model carefully (on paper) prior to entering it into the Arena<sup>TM</sup> environment. If you just sit down and try to program at the computer, the effort can result in confusing spaghetti code. You

should have a plan for defining your variables and use a naming convention for things that you use in the model. For example, attach an “R” to the end of your resource names or add a “v” to the beginning of your variables. Also, you should fill in the description property for your modules and use good common sense module names. In addition, you should list out the logic of your model in some sort of pseudo-code. Examples of pseudo-code were provided within the chapter. Additional examples of this will be given in future chapters of this text. Finally, you should use the submodel feature in Arena™ to organize your code into manageable and logically consistent pieces. You should treat simulation model development within Arena™ more like a programming effort than you might have first thought.

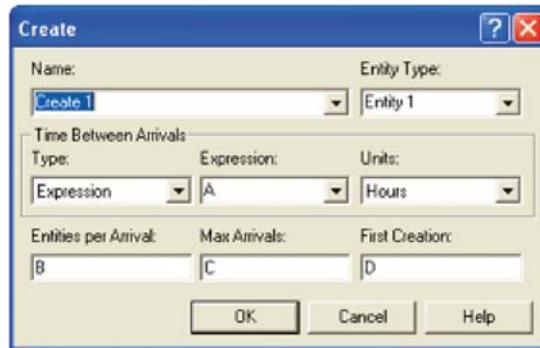
The next couple of chapters will build upon the modeling foundations learned in this chapter. Chapter 6 will concentrate on building models that incorporate randomness and how Arena™ facilitates modeling random processes. Then, in Chapter 7, you will learn how to analyze models that have randomness through the use of proper statistical techniques. Along the way, more of the modules and concepts needed to build more realistic simulation models in Arena™ will be presented.

## EXERCISES

- 5.1 The \_\_\_\_\_ of a system is defined to be that collection of variables necessary to describe the system at any time, relative to the objectives of study.
- 5.2 An \_\_\_\_\_ is defined as an instantaneous occurrence that may change the state of the system.
- 5.3 An \_\_\_\_\_ describes the properties of entities by data values.
- 5.4 The \_\_\_\_\_ attribute is a unique number assigned to an entity when it is created; however, if the entity is ever duplicated (cloned) in the model, the clones will have the same value for the attribute.
- 5.5 *True or False:* The Time Between option of the RECORD module will calculate and record the difference between a specified attribute’s value and the current simulation time.
- 5.6 Fill in the proper Arena module name for the given functionality.

Arena Module Name	Description of Module Functionality
	This module allows input and output to occur within the model.
	This module allows the user to define named logical/mathematical expressions that can be used throughout the model.
	This module allows entities to wait in queues until a user-specified matching criteria occurs.
	Used to provide alternative flow paths for an entity based on probabilistic or condition-based branching.
	Used to define a list of elements within Arena that can be indexed by the location in the list.
	Used to define different entity types for use within the model.
	Used to collect statistics on entities and remove the entities from the model once they have completed their activities within the model.

- 5.7 Groups of customers arrive to a Blues, Bikes, and BBQ T-shirt concession stand according to a Poisson process with a mean rate of 10 per hour. There is a 10% chance that a family of four will want T-shirts, a 30% chance that a family of three will want T-shirts, a 20% chance that a couple will want matching T-shirts, and a 40% chance that an individual person will want a T-shirt.



Specify expressions for A, B, C, and D in the above CREATE module to properly generate customers for the T-shirt stand.

- A: \_\_\_\_\_
- B: \_\_\_\_\_
- C: \_\_\_\_\_
- D: \_\_\_\_\_

- 5.8 Provide the missing information for steps in modeling:

- What is the system? What information is known by the system?
- What are the (a)\_\_\_\_\_? What information must be recorded for each (b)\_\_\_\_\_?
- What are the (c)\_\_\_\_\_?
- What are the (d)\_\_\_\_\_?
- Develop an (e)\_\_\_\_\_ for the life of the (f)\_\_\_\_\_
- Develop (g)\_\_\_\_\_ for the simulation model?

- 5.9 Fill in the proper Arena™ module name for the given functionality:

Arena Module Name	Description of Module Functionality
	This module allows input and output to occur within the model
	Used to collect statistics on entities and remove the entities from the model once they have completed their activities within the model
	Used to capture and tabulate statistics within the flowchart model area
	Used to create duplicates of an existing entity or to split a batched group of entities
	Used to make assignments to variables and attributes within the model

- 5.10 Suppose that the customers arriving to the drive through pharmacy can decide to enter the store instead of entering the drive through lane. Assume a 90% chance that the arriving customer decides to use the drive through pharmacy and a 10% chance that the customer decides to use the store. Model this situation with Arena<sup>TM</sup> and discuss the effect on the performance of the drive through lane. Run your model for 1 year, with 20 replications.
- 5.11 Suppose that a customer arriving to the drive through pharmacy will decide to balk if the number of cars waiting in line is 4 or more. A customer is said to *balk* if he or she refuses to enter the system and simply departs without receiving service. Model this situation using Arena<sup>TM</sup> and estimate the probability that a customer will balk because the line is too long. Run your model for 1 year, with 20 replications.
- 5.12 Consider a manufacturing system comprising two different machines and two operators. Each operator is assigned to run a single machine. Parts arrive with an exponentially distributed interarrival time with a mean of 3 minutes. The arriving parts are one of two types. Sixty percent of the arriving parts are Type 1 and are processed on Machine 1. These parts require the assigned operator for a 1-minute setup operation. The remaining 40% of the parts are Type 2 parts and are processed on Machine 2. These parts require the assigned operator for a 1.5-minute setup operation. The service times (excluding the setup time) are lognormally distributed with a mean of 4.5 minutes and a standard deviation of 1 minute for Type 1 parts and a mean of 7.5 minutes and a standard deviation of 1.5 minutes for Type 2 parts. Run your model for 20,000 minutes, with 20 replications. Report the utilization of the machines and operators. In addition, report the total time spent in the system for each type of part.
- 5.13 Incoming phone calls arrive according to a Poisson process with a rate of 1 call per hour. Each call is either for the accounting department or for the customer service department. There is a 30% chance that a call is for the accounting department and a 70% chance that the call is for the customer service department. The accounting department has one accountant available to answer the call, which typically lasts uniformly between 30 and 90 minutes. The customer service department has three operators that handle incoming calls. Each operator has their own queue. An incoming call designated for customer service is routed to operator 1, operator 2, or operator 3 with a 25%, 45%, and 30% chance, respectively. Operator 1 typically takes uniformly between 30 and 90 minutes to answer a call. The call-answering time of operator 2 is distributed according to a triangular distribution with a minimum of 30 minutes, a mode of 60 minutes, and a maximum of 90 minutes. Operator 3 typically takes exponential 60 minutes to answer a call. Run your simulation for 8 hours and estimate the average queue length for calls at the accountant, operator 1, operator 3, and operator 3. Develop an Arena<sup>TM</sup> simulation for this situation. Simulate 30 days of operation, where each day is 10 hours long. Report the utilization of the accountant and the operators as well as the average time that calls wait within the respective queues.
- 5.14 Write a program in Arena<sup>TM</sup> using iterative looping to compute the value of 7!
- 5.15 Samples of 20 parts from a metal grinding process are selected every hour. Typically 2% of the parts needs rework. Let  $X$  denote the number of parts in the sample of 20 that require rework. A process problem is suspected if  $X$  exceeds its mean by more than 3 standard deviations. Using Arena<sup>TM</sup>, simulate 30 hours of the process, that is,

30 samples of size 20, and estimate the chance that  $X$  exceeds its expected value by more than 3 standard deviations.

- 5.16 Samples of 20 parts from a metal grinding process are selected every hour. Typically 2% of the parts needs rework. Let  $X$  denote the number of parts in the sample of 20 that require rework. A process problem is suspected if  $X$  exceeds its mean by more than 3 standard deviations. Each time  $X$  exceeds its mean by more than 3 standard deviations all  $X$  of the parts requiring rework are sent to a rework station. Each part consists of two subcomponents, which are split off and repaired separately. The splitting process takes 1 worker and lasts  $U(10, 20)$  minutes per part. After the subcomponents have been split, they are repaired in different processes. Subcomponent 1 takes  $U(5, 10)$  minutes to repair with 1 worker at its repair process and subcomponent 2 takes  $\text{expo}(7.5)$  minutes to repair with one worker at its repair process. Once both the subcomponents have been repaired, they are joined back together to form the original part. The joining process takes 5 minutes with one worker. The part is then sent back to the main production area, which is outside the scope of this problem. Simulate 8 hours of production and estimate the average time that it takes a part to be repaired.
- 5.17 TV sets arrive at a two-inspector station for testing. The time between arrivals is exponential with a mean of 15 minutes. The inspection time per TV set is exponential with a mean of 10 minutes. On an average, 82% of the sets pass inspection. The remaining 18% is routed to an adjustment station with a single operator. Adjustment time per TV set is uniform between 7 and 14 minutes. After adjustments are made, sets are routed back to the inspection station to be retested. We are interested in estimating the total time a TV set spends in the system before it is released.

Develop an Arena model for this situation. Report the average system time for the TV sets based on 20 replications of 4800 minutes. Also report statistics for the average number of times a given job is adjusted.

- 5.18 A simple manufacturing system is staffed by three operators. Parts arrive according to a Poisson process with a mean rate of 2 per minute to a workstation for a drilling process at one of three identical drill presses. The parts wait in a single queue until a drill press is available. Each part has a particular number of holes that need to be drilled. Each hole takes a lognormal time to be drilled with an approximate mean of 1 minute and a standard deviation of 30 seconds. Once the holes are drilled, the part goes to the grinding operation. At the grinding operation, one of the three available operators grinds out the burrs on the part. This activity takes approximately 5 minutes plus or minus 30 seconds. After the grinding operation, the part leaves the system.

Develop an Arena model for this situation. Report the average system time for the parts based on 20 replications of 4800 minutes.

- 5.19 The Hog BBQ Joint is interested in understanding the flow of customers for dinner (5–9 pm). Customers arrive in parties of 2, 3, 4, or 5 with probabilities 0.4, 0.3, 0.2, or 0.1, respectively. The time between arrivals is exponentially distributed with a mean of 1.4 minutes. Customers must arrive prior to 9 pm in order to be seated. The dining area has 50 tables. Each table can seat two people. For parties, with more than two customers, the tables are moved together. Each arriving group gets in line to be seated. If there are already six parties in line, the arriving group will leave and go to another restaurant. The time that it takes to be served is triangularly distributed with parameters (14, 19, 24) in minutes. The time that it takes to eat is lognormally distributed with a

mean of 24 minutes and a standard deviation of 5 minutes. When customers are finished eating, they go to the cashier to pay their bill. The time that it takes the cashier to process the customers is gamma distributed with a mean of 1.5 minutes and a standard deviation of 0.5 minutes.

Develop an Arena™ model for this situation. Simulate 30 days of operation. Make a table like the following to summarize your results.

	Average	Half-width
Number of customers served		
Number of busy tables		
Number of waiting parties		
Number of parties that depart without eating		
Utilization of cashier		
Customer system time (in minutes)		
Probability of waiting to be seated >5 minutes		

- 5.20 In the tie-dye T-shirt model, the owner is expecting the business to grow during the summer season. The owner is interested in estimating the average time to produce an order and the utilization of the workers if the arrival rate for orders increases. Rerun the model for 30 eight-hour days with the arrival rate increased by 20%, 40%, 60%, and 80%. Will the system have trouble meeting the demand? Use the statistics produced by Arena™ to answer this question.
- 5.21 Suppose that the inspection and packaging process has been split into two processes for the tie-dye T-shirt system and assume that there is an additional worker to perform inspection. The inspection process is uniformly distributed between 2 and 5 minutes. After inspection, there is a 4% chance that the whole order will have to be scrapped (and redone). If the order fails inspection, the scrapped order should be counted and a new order should be initiated into the system. Hint: Consider redirecting the order back to the original SEPARATE module. If the order passes inspection, it goes to packaging where the packaging time is distributed according to a triangular distribution with parameters (2, 4, 10) all in minutes. Rerun the model for 30 eight-hour days, with the arrival rate increased by 20%, 40%, 60%, and 80%. Will the system have trouble meeting the demand? In other words, how does the throughput (number of shirts produced per day) change in response to the increasing demand rate?
- 5.22 Hungry customers arrive to a Mickey R's drive through restaurant at a mean rate of 10 per hour according to a Poisson process. Management is interested in improving the total time spent within the system (i.e., from arrival to departure with their food).

Management is considering a proposed system that splits the order taking, payment activity, and the order delivery processes. The first worker will take the orders from an order-taking speaker. This takes on an average 1 minute plus or minus 20 seconds uniformly distributed. When the order-taking activity is completed, the making of the order will start. It takes approximately 3 minutes ( $\pm 20$  seconds) to make the customer's order, uniformly distributed. Meanwhile, the customer will be instructed to drive to the first window to pay for the order. Assume that the time that it takes the customer to move forward is negligible. The first worker accepts the payment from the customer. This takes on an average  $45 \pm 20$  seconds uniformly distributed. After

paying for the order, the customer is instructed to pull forward to the second window, where a second worker delivers the order. Assume that the time that it takes the customer to move forward is negligible.

If the order is not completed by the time the customer reaches the second window, then the customer must wait for the order to be completed. If the order is completed before the customer arrives to the second window, then the order must wait for the customer. After both the order and the customer are at the second window, the second worker packages the customer's order and gives it to the customer. This takes approximately 30 seconds with a standard deviation of 10 seconds, lognormally distributed. After the customer receives their order, they depart.

Simulate this system for the period from 10 am to 2 pm. Report the total time spent in the system for the customers based on 30 days.

- 5.23 The city is considering improving its hazardous waste and bulk item drop off area to improve service. Cars arrive to the drop off area at a rate of 10 per hour according to a Poisson process. Each car contains items for drop off. There is a 10% chance that the car will contain 1 item, a 50% chance that the car will contain 2 items, and a 40% chance that the car will contain 3 items. There is an 80% chance that an item will be hazardous (e.g., chemicals, light bulbs, and electronic equipment) and a 20% chance that the item will be a bulk item, which cannot be picked up in the curbside recycling program. Of the 80% of items that have hazardous waste, about 10% is for electronic equipment that must be inspected and taken apart.

A single worker assists the citizen in taking the material out of their car and moving the material to the recycling center. This typically takes between 0.5 and 1.5 minutes per item (uniformly distributed) if the item is not a bulk item. If the item is a bulk item, then the time takes a minimum of 1 minute, most likely 2.5 minutes, with a maximum of 4 minutes per item triangularly distributed. The worker finishes all items in a car before processing the next car.

Another worker will begin sorting the items immediately after the item is unloaded. This process takes 1–2 minutes per item uniformly distributed. If the item is electronic equipment, the items are placed in front of a special disassembly station to be taken apart.

The same worker who performs sorting also performs the disassembly of the electronic parts. Items that require sorting take priority over items that require disassembly. Each electronic item takes between 8 and 16 minutes uniformly distributed to disassemble.

The hazardous waste recycling center is open for 7 hours per day, 5 days per week. Simulate 12 weeks of performance and estimate the following quantities:

- Utilization of the workers
- Average waiting time for items waiting to be unloaded
- Average number of items waiting to be unloaded
- Average number of items waiting to be sorted
- Average waiting time of items to be sorted
- Average number of items waiting to be disassembled
- Average waiting time for items waiting to be disassembled.

- 5.24 Orders for street lighting poles require the production of the tapered pole, the base assembly, and the wiring/lighting assembly package. Orders are released to the shop floor with an exponential time between arrival of 20 minutes. Assume that all the materials for the order are already available within the shop floor.

Once the order arrives, the production of the pole begins. Pole production requires that the sheet metal be cut to a trapezoidal shape. This process takes place on a cutting shear. After cutting, the pole is rolled using a press brake machine. This machine rolls the sheet to an almost closed form. After rolling, the pole is sealed on an automated welding machine. Each of these processes are uniformly distributed with ranges [3, 5], [6,10], and [4,8] minutes, respectively.

While the pole is being produced, the base is being prepared. The base is a square metal plate with four holes drilled for bolting the plate to the mounting piece and a large circular hole for attaching the pole to the base. The base plates are in stock so that only the holes need to be cut. This is done on a water jet cutting machine. This process takes approximately  $20 \pm 2$  minutes, triangularly distributed. After the holes are cut, the plate goes to a grinding/deburring station, which takes between 10 minutes, exponentially distributed.

Once the plate and the pole are completed, they are transported to the inspection station. Inspection takes 20 minutes, exponentially distributed with one operator. There could be a quality problem with the pole or the base (or both). The chance that the problem is with the base is 0.02, and the chance that the problem is with the pole is 0.01. If either or both have a quality issue, the pole and base go to a rework station for rework. Rework is performed by a single operator and typically takes between 100 minutes, exponentially distributed. After rework, the pole and base are sent to final assembly. If no problems occur with the pole or the base, the pole and base are sent directly to final assembly.

At the assembly station, the pole is fixed to the base plate and the wiring assembly is placed within the pole. This process takes 1 operator approximately 30 minutes with a standard deviation of 4 minutes according to a lognormal distribution. After assembly, the pole is sent to the shipping area for final delivery.

The shop is interested in taking on additional orders which would essentially double the arrival rate. Estimate the utilization of each resource and the average system time to produce an order for a lighting pole. Assume that the system runs 5 days per week, with 2 eight hours shifts per day. Any production that is not completed within 5 days is continued on the next available shift. Run the model for 10 years assuming 52 weeks per year to report your results.

- 5.25 Patients arrive at an emergency room where they are treated and then depart. Arrivals are exponentially distributed with a mean time between arrivals of 0.3 hours. Upon arrival, patients are assigned a rating of 1–5, depending on the severity of their ailments. Patients in Category 1 are the most severe and are immediately sent to a bed where they await medical attention. All other patients must first wait in the receiving room until a basic registration form and medical record are completed. They then proceed to a bed.

The emergency room has three beds, one registration nurse, and two doctors. In all cases, the priority for allocating these resources is based on the severity of the

ailment. Hint: Read the help for the QUEUE module and rank the queue by the severity attribute. The registration time for patients in Categories 2 through 5 is Uniform(0.1, 0.2) hours. The treatment time for all patients is triangularly distributed with the minimum, most likely, and maximum values differing according to the patient's category. The distribution of patients by category and the corresponding minimum, most likely, and maximum treatment times are summarized below.

Category	1	2	3	4	5
Percentage	6	8	18	33	35
Minimum	0.8	0.7	0.4	0.2	0.1
Most likely	1.2	0.95	0.6	0.45	0.35
Maximum	1.6	1.1	0.75	0.6	0.45

The required responses for this simulation include the following:

- Average number of patients waiting for registration
  - Utilization of beds
  - System time of each type of patient and overall (across patient types)
    - (a) Prepare an activity diagram for this problem
    - (b) Using a run length of 30 days, develop an Arena model to estimate the required responses. Report the responses based on estimating the system time of a patient regardless of type based on 50 replications.
- 5.26 Customers enter a fast-food restaurant according to an exponential interarrival time with a mean of 0.7 minutes (use stream 1). Customers have a choice of ordering one of three kinds of meals: (1) a soft drink, (2) fries, or (3) soft drink, fries, and a burger. Upon arrival to the restaurant, the customer enters a single queue, awaits the availability of a cashier, gives the order to the cashier, and then the customer pays the cashier. After the order is placed, the cooking can begin. The customer then waits until their order is ready. After receiving the order, the customer exits. A cashier may not take any additional orders until the current customer has paid. In this system, there are two cooks and two cashiers. The time to order and pay is represented by a triangular distribution with parameters (0.4, 0.8, 1.2) minutes and (0.2, 0.4, 0.6) minutes, respectively. The cooking time depends on the order as follows:

Type	Percentage, %	Cooking Time
1	30	Uniform(0.3,0.8)
2	15	Uniform(0.8,1.1)
3	55	Uniform(1.0, 1.4)

Model the system for 8 hours of operation with 30 replications. Make a table like the following to summarize your answers for your replications.

	Average	Half-Width
Type 1 throughput		
Type 2 throughput		
Type 3 throughput		
Utilization of cashiers		
Utilization of cooks		
Customer system time (in minutes)		
Customer waiting time (in minutes)		
Probability of wait > 5 minutes		



---

# 6

---

## MODELING RANDOMNESS IN SIMULATION

### LEARNING OBJECTIVES

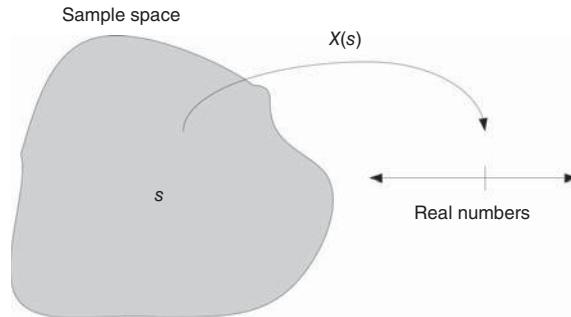
- To be able to model probability distributions based on input data.
- To be able to use Arena<sup>TM</sup> to model systems that contain randomness.

### 6.1 RANDOM VARIABLES AND PROBABILITY DISTRIBUTIONS

This section discusses some concepts in probability and statistics that are especially relevant to simulation. These will serve you well as you model randomness in the inputs of your simulation models. When an input process for a simulation is stochastic, you must develop a probabilistic model to characterize the process's behavior over time. Suppose that you are modeling the service times in the pharmacy example. Let  $X_i$  be a random variable that represents the service time of the  $i$ th customer. A random variable is a function that assigns a real number to each outcome,  $s$ , in a random process that has a set of possible outcomes,  $S$ .

In this case, the process is the service times of the customers and the outcomes are the possible values that the service times can take on, that is, the range of possible values for the service times.

The determination of the range of the random variable is part of the modeling process. For example, if the range of the service time random variable is the set of all possible positive real numbers, that is,  $X_i \in \mathbb{R}^+$  or in other words,  $X_i \geq 0$ , then the service time should be modeled as a *continuous* random variable (Figure 6.1).



**Figure 6.1** Random variables map outcomes to real numbers.

Suppose instead that the service time can only take on one of five discrete values 2, 5, 7, 8, and 10, then the service time random variable should be modeled as a *discrete* random variable. Thus, the first decision in modeling a stochastic input process is to appropriately define a random variable and its possible range of values.

The next decision is to characterize the probability distribution for the random variable (Figure 6.2). A probability distribution for a random variable is a function that maps from the range of the random variable to a real number,  $p \in [0, 1]$ . The value of  $p$  should be interpreted as the probability associated with the event represented by the random variable.

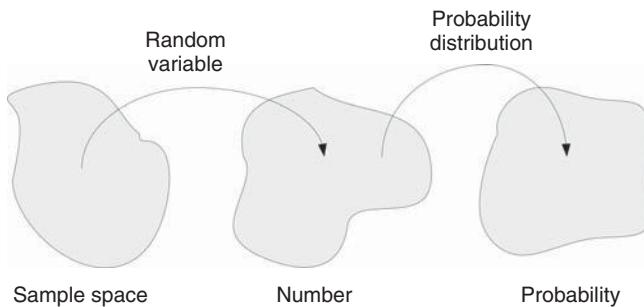
For a discrete random variable,  $X$ , with possible values  $x_1, x_2, \dots, x_n$  ( $n$  may be infinite), the function  $f(x)$  that assigned probabilities to each possible value of the random variable is called the probability mass function (PMF) and is denoted as

$$f(x_i) = P(X = x_i) \quad (6.1)$$

where  $f(x_i) \geq 0$  for all  $x_i$  and  $\sum_{i=1}^n f(x_i) = 1$ . The PMF describes the probability value associated with each discrete value of the random variable.

For a continuous random variable,  $X$ , the mapping from real numbers to probability values is governed by a probability density function (PDF)  $f(x)$  and has the following properties:

1.  $f(x) \geq 0$
2.  $\int_{-\infty}^{\infty} f(x)dx = 1$  (The area must sum to 1.)



**Figure 6.2** Probability distributions map random variables to probabilities.

$$3. P(a \leq x \leq b) = \int_a^b f(x)dx \text{ (The area under } f(x) \text{ between } a \text{ and } b.)$$

The PDF describes the probability associated with a range of possible values for a continuous random variable.

A cumulative distribution function (CDF) for a discrete or continuous random variable can also be defined. For a discrete random variable, the CDF is defined as

$$F(x) = P(X \leq x) = \sum_{x_i \leq x} f(x_i) \quad (6.2)$$

and satisfies  $0 \leq F(x) \leq 1$  and for  $x \leq y, F(x) \leq F(y)$ .

The CDF of a continuous random variable is

$$F(x) = P(X \leq x) = \int_{-\infty}^x f(u)du \text{ for } -\infty < x < \infty \quad (6.3)$$

Thus, when modeling the elements of a simulation model that have randomness, one must determine the following:

- Whether or not the randomness is discrete or continuous.
- The form of the distribution function (i.e., the PMF or PDF).

To develop an understanding of the probability distribution for the random variable, it is useful to characterize various properties of the distribution such as the expected value and variance of the random variable.

The *expected value* of a discrete random variable  $X$  is denoted by  $E[X]$  and is defined as

$$E[X] = \sum_x xf(x) \quad (6.4)$$

where the sum is defined through all possible values of  $x$ . The *variance* of  $X$  is denoted by  $\text{Var}[X]$  and is defined as

$$\begin{aligned} \text{Var}[X] &= E[(X - E[X])^2] \\ &= \sum_x (x - E[X])^2 f(x) \\ &= \sum_x x^2 f(x) - (E[X])^2 \\ &= E[X^2] - (E[X])^2 \end{aligned} \quad (6.5)$$

Suppose  $X$  is a continuous random variable with PDF,  $f(x)$ , then the expected value of  $X$  is

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx \quad (6.6)$$

and the variance of  $X$  is

$$\text{Var}[X] = \int_{-\infty}^{\infty} (x - E[X])^2 f(x)dx = \int_{-\infty}^{\infty} x^2 f(x)dx - (E[X])^2 \quad (6.7)$$

which is equivalent to  $\text{Var}[X] = E[X^2] - (E[X])^2$  where

$$E[X^2] = \int_{-\infty}^{\infty} x^2 f(x) dx$$

Another parameter that is often useful is the coefficient of variation. The coefficient of variation is defined as

$$c_v = \frac{\sqrt{\text{Var}[X]}}{E[X]} \quad (6.8)$$

The coefficient of variation measures the amount of variation relative to the mean value (provided that  $E[X] \neq 0$ ).

To estimate  $E[X]$ , the sample average,  $\bar{X}(n)$ ,

$$\bar{X}(n) = \frac{1}{n} \sum_{i=1}^n X_i \quad (6.9)$$

is often used. To estimate  $\text{Var}[X]$ , assuming that the data are independent, the sample variance,  $S^2$ ,

$$S^2(n) = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2 \quad (6.10)$$

can be used. Thus, an estimator for the coefficient of variation is

$$\hat{c}_v = \frac{s}{\bar{x}} \quad (6.11)$$

A number of other statistical quantities are also useful when trying to characterize the properties of a distribution:

- *Skewness.* Measures the asymmetry of the distribution about its mean.
- *Kurtosis.* Measures the degree of peakedness of the distribution.
- *Order Statistics.* Used when comparing the sample data to the theoretical distribution via P–P plots or Q–Q plots.
- *Quantiles (First Quartile, Median, Third Quartile).* Summarizes the distribution of the data.
- *Minimum, Maximum, and Range.* Indicates the range of possible values.

Skewness can be estimated by

$$\hat{\gamma}_1 = \frac{\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^3}{[S^2]^{3/2}} \quad (6.12)$$

For a unimodal distribution, negative skew indicates that the tail on the left side of the PDF is longer or fatter than the right side. Positive skew indicates that the tail on the right side is longer or fatter than the left side. A value of skewness near zero indicates symmetry.

Kurtosis can be estimated by

$$\hat{\gamma}_2 = \frac{n-1}{(n-2)(n-3)}((n+1)g_2 + 6) \quad (6.13)$$

where  $g_2$  is

$$g_2 = \frac{\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^4}{\left(\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2\right)^2} - 3 \quad (6.14)$$

Order statistics are just a fancy name for the sorted data. Let  $x_1, x_2, \dots, x_n$  represent a sample of data. If the data is sorted from smallest to largest, then the  $i$ th ordered element can be denoted as  $x_{(i)}$ . For example,  $x_{(1)}$  is the smallest element and  $x_{(n)}$  is the largest, so that  $x_{(1)}, x_{(2)}, \dots, x_{(n)}$  represents the ordered data and these values are called the order statistics. From the order statistics, a variety of other statistics can be computed:

1. Minimum =  $x_{(1)}$
2. Maximum =  $x_{(n)}$
3. Range =  $x_{(n)} - x_{(1)}$

The median,  $\tilde{x}$ , is a measure of central tendency such that one-half of the data is above it and one-half of the data is below it. The median can be estimated as follows:

$$\tilde{x} = \begin{cases} x_{((n+1)/2)} & n \text{ is odd} \\ \frac{x_{(n/2)} + x_{((n/2)+1)}}{2} & n \text{ is even} \end{cases} \quad (6.15)$$

For example, consider the following data:

$$x_{(1)} = 3, x_{(2)} = 5, x_{(3)} = 7, x_{(4)} = 7, x_{(5)} = 38$$

Because  $n = 5$ , we have

$$\begin{aligned} \frac{n+1}{2} &= \frac{5+1}{2} = 3 \\ \tilde{x} &= x_{(3)} = 7 \end{aligned}$$

Suppose we have the following data:

$$x_{(1)} = 3, x_{(2)} = 5, x_{(3)} = 7, x_{(4)} = 7$$

Because  $n = 4$ , we have

$$\begin{aligned} x_{(n/2)} &= x_{(2)} \\ x_{((n/2)+1)} &= x_{(3)} \\ \tilde{x} &= \frac{x_{(2)} + x_{(3)}}{2} = \frac{5+7}{2} = 6 \end{aligned}$$

The first quartile is the first 25% of the data and can be thought of as the “median” of the first half of the data. Similarly, the third quartile is the first 75% of the data or the “median” of the second half of the data. Different methods are used to estimate these quantities in various software packages; however, their interpretation is the same, summarizing the distribution of the data.

## 6.2 MODELING WITH DISCRETE DISTRIBUTIONS

There are a wide variety of discrete random variables that often occur in simulation modeling. Appendix A summarizes some common discrete distributions. In this section, we outline some of their common uses.

A Bernoulli distribution models the behavior of a random variable that follows a Bernoulli trial. A Bernoulli trial has the following characteristics:

- The result of each trial may be considered either a success or a failure, that is, only two outcomes are possible.
- The probability,  $p$ , of success is the same in every trial.
- The trials are independent. That is, trials have no influence over each other or memory of each other.

A Bernoulli distribution is often used to model a flow process in which an entity can take two possible paths at a decision point. For example, if a part is inspected and the chance that the part is defective is  $p$  and each part is independent of any other part, then whether or not the part is defective can be characterized by a Bernoulli distribution.

The DECIDE module easily models this situation with the two-way by chance option using  $p\%$  in the chance amount field. Often, an attribute associated with the entity must be set based on a Bernoulli trial. For example, suppose a part either requires grinding or does not with probability  $p$  representing the probability of required grinding. Assuming that 1 = grinding required and 0 = grinding not required, we can model this situation with an ASSIGN module and a DISC() distribution as follows.

```
ASSIGN: myGrindingFlag = DISC(p, 1, 1.0, 0)
```

The DISC() distribution function with two outcomes 1 and 0 is used here to model a Bernoulli random variable.

As discussed in Chapter 2 by using a Bernoulli random variable, we can easily model situations involving binomial, geometric, and negative binomial distributions. A binomial distribution models a random variable that represents the number of successes in  $n$  Bernoulli trials where  $Y_i$  represents the outcome of the  $i$ th trial. Since a success is indicated with 1, the number of successes is simply the sum of the  $Y$ 's,  $X = \sum_{i=1}^n Y_i$ . The random variable,  $X$ , will have a binomial distribution with parameters  $n$  for the number of trials and  $p$  for the probability of success for each trial. For example, suppose 10 parts are to be produced and assume that whether or not each part is defective is a Bernoulli trial, then the number of defective parts out of the 10 parts that are defective is characterized by a binomial distribution with parameters  $n = 10$  and  $p$ . Simulation often has situations involving binomial random variables. The simplest way to simulate a binomial random variable is to first simulate each Bernoulli trial and then count the number of successes. This can be readily accomplished using a while-loop.

A geometric distribution models a random variable that represents the number of trials until the first success in a sequence of Bernoulli trials. Suppose that a repair machine is available to rework parts from a machining center. In addition, suppose that the repair machine is only set up after the first defective part is produced. Then, the number of parts until the repair machine is set up for the first time can be modeled by a geometric distribution. Generating a shifted geometric random variable in Arena<sup>TM</sup> can be accomplished with the following formula:

$$1 + \text{AINT}(\text{LN}(1 - \text{UNIF}(0, 1)) / \text{LN}(1 - p)) \quad (6.16)$$

The geometric distribution is a special case of the negative binomial distribution. Suppose now that instead of setting up the repair machine when the first defective part is produced that the system waits until four defective parts are produced. Then, the number of parts produced until the repair machine is set up can be modeled by a negative binomial distribution. That is, if  $X$  denotes a random variable that represents the number of Bernoulli trials until  $r$  successes occur, then  $X$  has a negative binomial distribution with parameters  $r$  and  $p$ . Generation of a negative binomial random variable in Arena<sup>TM</sup> can be accomplished by using a loop that stops when the  $r$ th success occurs. Alternatively, the formula in Equation (6.16) can be used within a while-loop construct to implement the convolution relationship between geometric random variables and negative binomial random variables as discussed in Chapter 2. In other words, the sum of  $r$  geometric random variables with success probability  $p$  results in a negative binomial random variable with parameters  $r$  and  $p$ .

The discrete uniform distribution can be used to model situations where there are a finite discrete set of outcomes that have equal probability of occurring (Table 6.1). There are two basic definitions for this distribution. The first definition is defined over a consecutive set of integers on the range  $[a, b]$ . Suppose the parts are numbered in a queue waiting to be inspected. Each part is denoted by its rank in the queue, for example, 1 = first in queue, ..., 10 = last in queue. Then, a part can be randomly selected for inspection by using a discrete uniform distribution over the range from  $a = 1$  to  $b = 10$ . This is easy to implement within Arena<sup>TM</sup> using any of the following expressions:

- $a + \text{AINT}((b - a + 1)\text{RA})$
- $\text{AINT}(\text{UNIF}(a, b) + 0.5)$
- $\text{ANINT}(\text{UNIF}(a, b))$

The second definition of a discrete uniform random variable allows for the set of possible outcomes to be over an arbitrary set of integer values. The DISC() distribution is designed to model this situation. Please see Appendix A for the properties of these distributions.

The Poisson distribution is often used to model the number of events within an interval of time, for example, the number of phone calls to a call center in an hour, the number of persons arriving to a bank, the number of cars arriving to an intersection in an hour, and the number of demands for inventory within a month. In addition, it is used to model the number of defects that occur within a length (or volume) of an item, for example, the number of typos in a book or the number of pot holes in a mile of road.

Consider an interval of real numbers and assume that incidents occur at random throughout the interval. The generation of Poisson random variables was discussed in Example 2.10. The Poisson distribution is intimately related to the exponential distribution

**TABLE 6.1 Common Modeling Situations for Discrete Distributions**

Distribution	Modeling Situations
Bernoulli( $p$ )	Independent trials with success probability $p$
Binomial( $n,p$ )	Sum of $n$ Bernoulli trials with success probability $p$
Geometric( $p$ )	Number of Bernoulli trials until the first success
Negative Binomial( $r,p$ )	Number of Bernoulli trials until the $r$ th success
Discrete Uniform( $a,b$ )	Equally likely outcomes over range $(a, b)$
Discrete Uniform $v_1, \dots, v_n$	Equally likely over values $v_i$
Poisson( $\lambda$ )	Counts of occurrences in an interval, an area, or a volume

as discussed in Chapter 2. The following function in Arena™ can be used to generate from a Poisson distribution: *POISSON(mean)*.

The Arena™ users guide has an appendix that describes the available distributions. These can also be found by searching on *Distributions* within Arena's Help system.

### 6.3 MODELING WITH CONTINUOUS DISTRIBUTIONS

Continuous distributions model can be used to model situations where the set of possible values occurs in an interval or set of intervals. Within discrete-event simulation, the most common use of continuous distributions is for the modeling of the time to perform a task. Appendix A summarizes the properties of common continuous distributions.

The continuous uniform distribution can be used to model situations in which you have a lack of data and it is reasonable to assume that everything is equally likely within an interval. The uniform distribution is also commonly used to model machine processing times that have very precise time intervals for completion. The expected value and variance of a random variable with a continuous uniform distribution over the interval  $(a, b)$  is

$$\begin{aligned} E [X] &= \frac{a + b}{2} \\ \text{Var } [X] &= \frac{(b - a)^2}{12} \end{aligned}$$

Often the continuous uniform distribution is specified by indicating the  $\pm$  around the expected value. For example, we can say that a continuous uniform over the range  $(5, 10)$  is the same as a uniform with  $7.5 \pm 2.5$ . The uniform distribution is symmetric over its defined interval.

The triangular distribution is also useful in situations with a lack of data if you can characterize a most likely value for the random variable in addition to its range (minimum and maximum). This makes the triangular distribution very useful when the only data that you might have on task times comes from interviewing people. It is relatively easy for someone to specify the most likely task time, a minimum task time, and a maximum task time. You can create a survey instrument that asks multiple people familiar with the task to provide these three estimates. From the survey, you can average the responses to develop an approximate model. This is only one possibility for how to combine the survey values.

If the most likely value is equal to one-half the range, then the triangular distribution is symmetric. In other words, 50% of the data is above and below the most likely value. If the most likely value is closer to the minimum value, then the triangular distribution is right

skewed (more area to the right). If the most likely value is closer to the maximum value, then the triangular distribution is left-skewed. The ability to control the skewness of the distribution in this manner also makes this distribution attractive.

The beta distribution can also be used to model situations where there is a lack of data. It is a bounded continuous distribution over the range from (0, 1) but can take on a wide variety of shapes and skewness characteristics. The beta distribution has been used to model the task times on activity networks and for modeling uncertainty concerning the probability parameter of a discrete distribution, such as the binomial. The beta distribution is commonly shifted to be over a range of values  $(a, b)$ , see Section 2.5.4.

The exponential distribution is commonly used to model the time between events. Often, when only a mean value is available (from the data or from a guess), the exponential distribution can be used. A random variable,  $X$ , with an exponential distribution rate parameter  $\lambda$  has

$$\begin{aligned} E [X] &= \frac{1}{\lambda} \\ \text{Var } [X] &= \frac{1}{\lambda^2} \end{aligned}$$

Notice that the variance is the square of the expected value. This is considered to be highly variable. The coefficient of variation for the exponential distribution is  $c_v = 1$ . Thus, if the coefficient of variation estimated from the data has a value near 1.0, then an exponential distribution may be a possible choice for modeling the situation.

An important property of the exponential distribution is the lack of memory property. The lack of memory property of the exponential distribution states that given  $\Delta t$  is the time period that elapsed since the occurrence of the last event, the time  $t$  remaining until the occurrence of the next event is independent of  $\Delta t$ . This implies that  $P\{X > \Delta t + t | X > t\} = P\{X > t\}$ . This property indicates that the probability of the occurrence of the next event is dependent upon the length of the interval since the last event, but not the absolute time of the last occurrence. It is the interval of elapsed time that matters. In a sense, the process's clock resets at each event time and the past does not matter when predicting the future. Thus, it "forgets" the past. This property has some very important implications, especially when modeling the time to failure. In most situations, the history of the process does matter (such as wear and tear on the machine). In which case, the exponential distribution may not be appropriate. Other distributions of the exponential family may be more useful in these situations such as the gamma and Weibull distributions. Why is the exponential distribution often used? There are two reasons: (i) often it is a good model for many situations found in nature and (ii) it has very convenient mathematical properties.

While the normal distribution is a mainstay of probability and statistics, you need to be careful when using it as a distribution for input models because it is defined over the entire range of real numbers. For example, within simulation, the time to perform a task is often required; however, time must be a positive real number. Clearly, since a normal distribution can have negative values, using a normal distribution to model task times can be problematic. If you attempt to delay for negative time you will receive an error. Instead of using a normal distribution, you might use a truncated normal, see Section 2.5.4. Alternatively, you can choose from any of the distributions that are defined on the range of positive real numbers, such as the lognormal, gamma, Weibull, and exponential distributions. The lognormal distribution is a convenient choice because it is also specified by two parameters: the mean and variance.

**TABLE 6.2 Common Modeling Situations for Continuous Distributions**

Distribution	Modeling Situations
Uniform	When you have no data, everything is equally likely to occur within an interval, machine task times
Normal	Modeling errors; modeling measurements, length, etc.; modeling the sum of a large number of other random variables
Exponential	Time to perform a task, time between failures, distance between defects
Erlang	Service times, multiple phases of service with each phase exponential
Weibull	Time to failure, time to complete a task
Gamma	Repair times, time to complete a task, replenishment lead time
Lognormal	Time to perform a task, quantities that are the product of a large number of other quantities
Triangular	Rough model in the absence of data assume a minimum, a maximum, and a most likely value
Beta	Useful for modeling task times on bounded range with little data, modeling probability as a random variable

Table 6.2 lists these modeling situation as well as others for various common continuous distributions.

Once we have a good idea about the type of random variable (discrete or continuous) and some ideas about the distribution of the random variable, the next step is to fit a distributional model to the data. This is called input distribution modeling.

## 6.4 INPUT DISTRIBUTION MODELING

When performing a simulation study, there is no substitution for actually observing the system and collecting the data required for the modeling effort. As outlined in Section 1.7, a good simulation methodology recognizes that modeling and data collection often occurs in parallel. That is, observing the system allows conceptual modeling which allows for an understanding of the input models that are needed for the simulation. The collection of the data for the input models allow further observation of the system and further refinement of the conceptual model, including the identification of additional input models. Eventually, this cycle converges to the point where the modeler has a well-defined understanding of the input data requirements. The data for the input model must be collected and modeled.

Input modeling begins with data collection, probability, statistics, and analysis. There are many methods available for collecting data, including time study analysis, work sampling, historical records, and automatically collected data. Time study and work sampling methods are covered in a standard industrial engineering curriculum. Observing the time an operator takes to perform a task via a time study results in a set of observations of the task times. Hopefully, there will be sufficient observations for applying the techniques discussed in this section.

Work sampling is useful for developing the percentage of time associated with various activities. This sort of study can be useful in identifying probabilities associated with performing tasks and for validating the output from the simulation models. Historical

records and automatically collected data hold promise for allowing more data to be collected and also pose difficulties related to the quality of the data collected. In any of the above mentioned methods, the input models will only be as good as the data and processes used to collect the data.

One especially important caveat for new simulation practitioners: do not rely on the people in the system you are modeling to correctly collect the data for you. If you do rely on them to collect the data, you must develop documents that clearly define what data is needed and how to collect the data. In addition, you should train them to collect the data using the methods that you have documented. Only through careful instruction and control of the data collection processes will you have confidence in your input modeling.

A typical input modeling process includes the following procedures:

1. *Documenting the process being modeled.* Describe the process being modeled and define the random variable to be collected. When collecting task times, you should pay careful attention to clearly define when the task starts and when the task ends. You should also document what triggers the task.
2. *Developing a plan for collecting the data and then collect the data.* Develop a sampling plan, describe how to collect the data, perform a pilot run of your plan, and then collect the data.
3. *Graphical and statistical analyses of the data* Using standard statistical analysis tools, you should visually examine your data. This should include plots such as a histogram, a time series plot, and an autocorrelation plot. Again, using statistical analysis tools, you should summarize the basic statistical properties of the data, for example, sample average, sample variance, minimum, maximum, and quartiles.
4. *Hypothesizing distributions.* Using what you have learned from steps 1–3, you should hypothesize possible distributions for the data.
5. *Estimating parameters.* Once you have possible distributions in mind, you need to estimate the parameters of those distributions so that you can analyze whether the distribution provides a good model for the data. With current software, this step as well as steps 3, 4, and 6 have been largely automated.
6. *Checking goodness of fit for hypothesized distributions.* In this step, you should assess whether or not the hypothesized probability distributions provide a good fit for the data. This should be done both graphically (e.g., histograms, P–P plots, and Q–Q plots) and via statistical tests (e.g., chi-squared test and Kolmogorov–Smirnov (K–S) test). As a part of this step, you should perform some sensitivity analysis on your fitted model.

During the input modeling process and after it is completed, you should document your process. This is important for two reasons. First, much can be learned about a system simply by collecting and analyzing data. Second, in order to have your simulation model accepted as useful by decision makers, they must *believe* in the input models. Even non-simulation savvy decision makers understand the old adage “Garbage In = Garbage Out”. The documentation helps to build credibility and allows you to illustrate how the data was collected.

The following sections illustrate the input modeling process with concrete examples.

## 6.5 FITTING DISCRETE DISTRIBUTIONS

This section illustrates how to model and fit a discrete distribution to data. Although the steps in modeling discrete and continuous distributions are very similar, the processes and tools utilized vary somewhat. The first thing to truly understand is the difference between discrete and continuous random variables. Discrete distributions are used to model discrete random variables. Continuous distributions are used to model continuous random variables. This may seem obvious but it is a key source of confusion for novice modelers.

Discrete random variables take on any of a specified finite or countable list of values. Continuous random variables take on any numerical value in an interval or collection of intervals. The source of confusion is when looking at a file of the data, you might not be able to tell the difference. The discrete values may have decimal places and then the modeler thinks that the data is continuous. The modeling starts with what is being collected and how it is being collected (not with looking at a file!).

### 6.5.1 Fitting a Poisson Distribution

Since the Poisson distribution is very important in simulation modeling, the discrete input modeling process will be illustrated by fitting a Poisson distribution. Example 6.1 presents data collected from an arrival process. As noted in Table 6.2, the Poisson distribution is a prime candidate for modeling this type of data.

#### EXAMPLE 6.1 Modeling Arrival Data with a Poisson Distribution

Suppose that we are interested in modeling the demand for a computer laboratory during the morning hours between 9 am and 11 am on normal weekdays. During this time, a team of undergraduate students has collected the number of students arriving to the computer laboratory during 15-minute intervals over a period of 4 weeks. The observations per 15-minute interval are overviewed in Table 6.3. The full data set is available with the chapter files.

Since there are four 15-minute intervals in each hour for each 2-hour time period, there are eight observations per day. Since there are 5 days per week, we have 40 observations per week for a total of  $40 \times 4 = 160$  observations. Check whether a Poisson distribution is an appropriate model for this data.

The solution to Example 6.1 involves the following steps:

1. Visualize the data.
2. Check if the week, period, or day of week influence the statistical properties of the count data.
3. Tabulate the frequency of the count data.
4. Estimate the mean rate parameter of the hypothesized Poisson distribution.
5. Perform goodness-of-fit tests to test the hypothesis that the number of arrivals per 15-minute interval has a Poisson distribution versus the alternative that it does not have a Poisson distribution.

**TABLE 6.3 Computer Laboratory Arrival Counts by Week, Period, and Day**

Week	Period	M	T	W	TH	F
1	9:00–9:15 am	8	5	16	7	7
1	9:15–9:30 am	8	4	9	8	6
1	9:30–9:45 am	9	5	6	6	5
1	9:45–10:00 am	10	11	12	10	12
1	10:00–10:15 am	6	7	14	9	3
1	10:15–10:30 am	11	8	7	7	11
1	10:30–10:45 am	12	7	8	3	6
1	10:45–11:00 am	8	9	9	8	6
2	9:00–9:15 am	10	13	7	7	7
:	:	:	:	:	:	:
3	9:00–9:15 am	5	7	14	8	8
:	:	:	:	:	:	:
4	9:00–9:15 am	7	11	8	5	4
:	:	:	:	:	:	:
4	10:45–11:00 am	8	9	7	9	6

### 6.5.2 Visualizing the Data

When analyzing a data set, it is best to begin with visualizing the data. We will analyze this data utilizing the R statistical software package. Assuming that the data is in a comma separate value (csv) file called *PoissonCountData.csv* in the R working directory, the following commands will read the data into the R environment, plot a histogram, plot a time series plot, and make an autocorrelation plot of the data.

```
> p2 = read.csv("PoissonCountData.csv")
> hist(p2$N, main="Computer Lab Arrivals", xlab = "Counts")
> plot(p2$N,type="b",main="Computer Lab Arrivals", ylab = "Count",
+       xlab = "Observation#")
> acf(p2$N, main = "ACF Plot for Computer Lab Arrivals")
```

Figure 6.3 illustrates the layout of the data frame in R. The first column indicates the week, the second column represents the period of the day, the third column indicates the day of the week, and the last column indicated with the variable, *N*, represents the count for the week, period, day combination. To access a particular column within the data frame, you use the \$ operator. Thus, the reference, *p2\$N*, accesses all the counts across all of the week, period, and day combinations. The variable, *p2\$N*, is subsequently used in the *hist*, *plot*, and *acf* commands.

As can be seen in Figure 6.4, the data has a somewhat symmetric shape with nothing unusual appearing in the figure. The time series plot, Figure 6.5, illustrates no significant pattern (e.g., trends). Finally, the autocorrelation plot, Figure 6.6, shows no significant correlation with respect to observation number.

Because arrival data often varies with time, it would be useful to examine whether or not the count data depends in some manner on when it was collected. For example, perhaps the computer lab is less busy on Fridays. In other words, the counts may depend on the day of the week. We can test for dependence on various factors by using a chi-square-based contingency table test. These tests are summarized in introductory statistics books. See Montgomery and Runger [2006].

R data frame of computer laboratory arrivals.

	Week	Period	Day	N
1	1	1	M	8
2	1	2	M	8
3	1	3	M	9
4	1	4	M	10
5	1	5	M	6
6	1	6	M	11
7	1	7	M	12
8	1	8	M	8
9	2	1	M	10
10	2	2	M	6

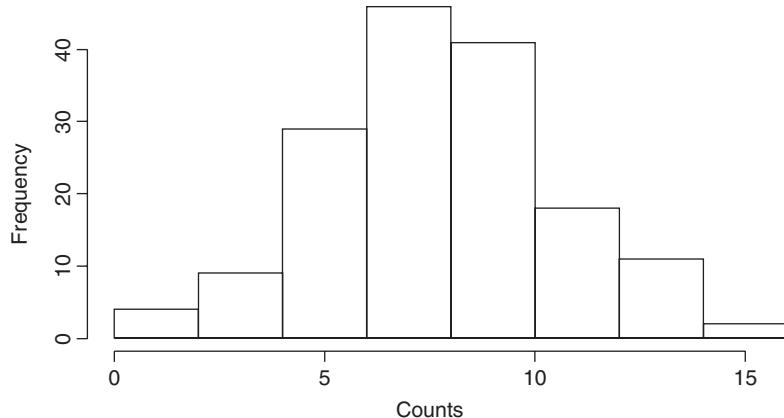
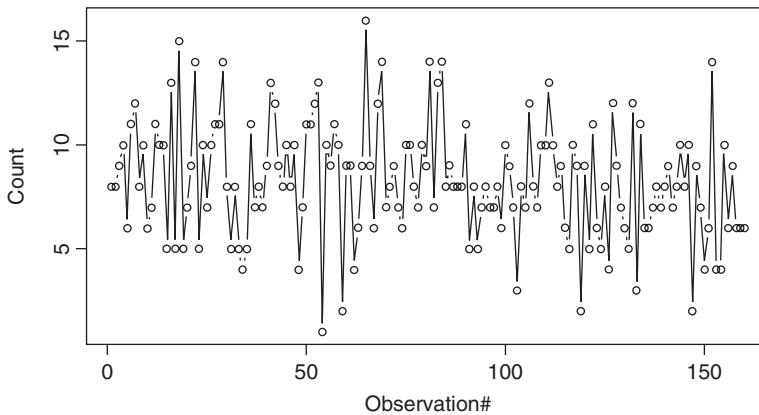


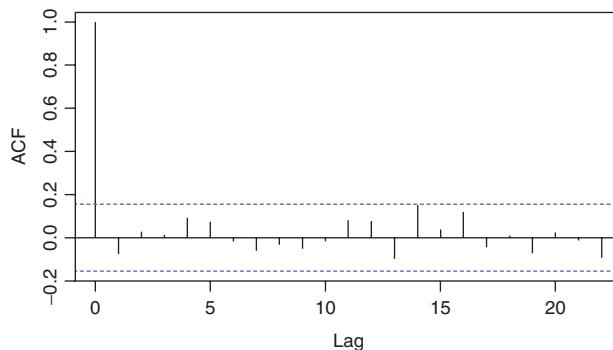
Figure 6.4 Histogram of computer laboratory arrivals.

First, we can try to visualize any patterns based on the factors. We can do this easily with a scatter plot matrix within the *lattice* package of R. In addition, we can use the *xtabs* function to tabulate the data by week, period, and day. The *xtabs* function specifies a modeling relationship between the observations and factors. In the R listing,  $N \sim Week + Period + Day$  indicates that we believe that the count column  $N$  in the data, depends on the *Week*, *Period*, and the *Day*. This builds a statistical object that can be summarized using the *summary* command.

```
> library(lattice)
> splom(p2)
> mytable = xtabs(N~Week + Period + Day, data=p2)
> summary(mytable)
Call: xtabs(formula = N ~ Week + Period + Day, data = p2)
```



**Figure 6.5** Time series plot of computer laboratory arrivals.



**Figure 6.6** ACF plot for computer laboratory arrivals.

Number of cases in table: 1324

Number of factors: 3

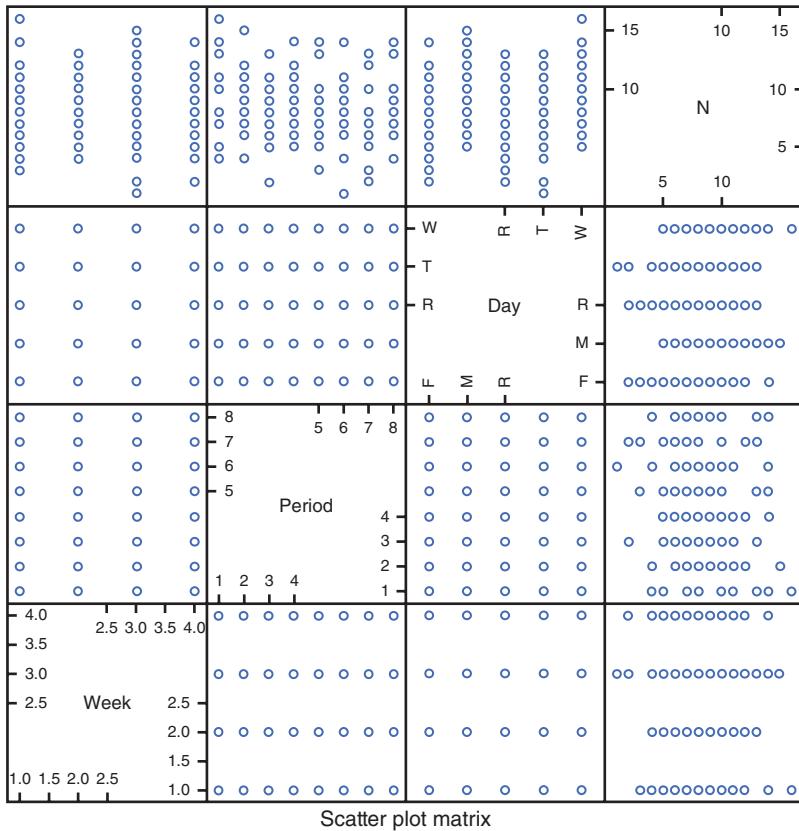
Test for independence of all factors:

Chisq = 133.76, df = 145, p-value = 0.7384

A scatter plot matrix plots the variables in a matrix format that makes it easier to view pairwise relationships within the data. Figure 6.7 presents the results of the scatter plot. Within the cells, the data looks reasonably “uniform.” That is, there are no discernible patterns to be found. This provides evidence that there is not likely to be dependence between these factors. To formally test this hypothesis, we can use the multifactor contingency table test provided by using the *summary* command on the output object, *myTable* of the *xtabs* command.

### 6.5.3 Statistical Analysis of the Data

The results of using *summary(mytable)* show that the chi-square test statistic has a very high *p*-value, 0.7384, when testing if *N* depends on *Week*, *Period*, and *Day*. The null hypothesis,  $H_0$ , of a contingency table test of this form states that the counts are independent of the factors versus the alternative,  $H_a$ , that the counts are dependent on the factors. Since the



**Figure 6.7** Scatter plot matrix from lattice package for computer laboratory arrival counts.

*p*-value is very high, we should not reject  $H_0$ . What does this all mean? In essence, we can now treat the arrivals as 160 independent observations. Thus, we can proceed with formally testing if the counts come from a Poisson distribution without worrying about time or factor dependencies within the data. If the results of the analysis indicated dependence on the factors, then we might need to fit separate distributions based on the dependence. For example, if we concluded that the days of the week were different (but the week and period did not matter), then we could try to fit a separate Poisson distribution for each day of the week. When the mean rate of occurrence depends on time, this situation warrants the investigation of using a nonhomogeneous (non-stationary) Poisson process. The estimation of the parameters of a non-stationary Poisson process is beyond the scope of this text. The interested reader should refer to Leemis [1991] and other such references.

Testing if the Poisson distribution is a good model for this data can be accomplished using the chi-squared goodness-of-fit test for a Poisson distribution. Section 2.4.1.1 first presented the theory of the chi-squared goodness-of-fit testing within the context of testing pseudorandom numbers. The basic procedure is the same here, except in this instance, the distribution being tested is the Poisson distribution rather than the uniform distribution. The basic approach is to compare the hypothesized distribution function in the form of the PDF, PMF, or the CDF to a fit of the data. This implies that you have hypothesized a distribution

and estimated the parameters of the distribution in order to compare the hypothesized distribution to the data. For example, suppose that you hypothesize that the Poisson distribution will be a good model for the count data. Then, you need to estimate the rate parameter associated with the Poisson distribution.

There are two main methods for estimating the parameters of distribution functions: (i) the method of moments and (ii) the method of maximum likelihood. The method of moments matches the empirical moments to the theoretical moments of the distribution and attempts to solve the resulting system of equations. The method of maximum likelihood attempts to find the parameter values that maximize the joint probability distribution function of the sample. Estimation of the parameters from sample data is based on important statistical theory that requires the estimators for the parameters to satisfy statistical properties (e.g., unique, unbiased, invariant, and consistency). It is beyond the scope of this book to cover the properties of these techniques. The interested reader is referred to Law [2007] or to Casella and Berger [1990] for more details on the theory of these methods.

To make concrete the challenges associated with fitting the parameters of a hypothesized distribution, the maximum likelihood method for fitting the Poisson distribution will be used on the count data. Suppose that you hypothesize that the distribution of the count of the number of arrivals in the  $i$ th 15-minute interval can be modeled with a Poisson distribution. Let  $N$  be the number of arrivals in a 15-minute interval. Note that the intervals do not overlap and that we have shown that they can be considered independent of each other. We are hypothesizing that  $N$  has a Poisson distribution with rate parameter  $\lambda$ , where  $\lambda$  represents the expected number of arrivals per 15-minute interval.

$$f(n; \lambda) = P\{N = n\} = \frac{e^{-\lambda}(\lambda)^n}{n!} \quad (6.17)$$

Let  $N_i$  be the number of arrivals in the  $i$ th interval of the  $k = 160$  intervals. The  $N_1, N_2, \dots, N_k$  form a random sample of size  $k$ . Then, the joint PMF of the sample is

$$L(\lambda) = g(n_1, n_2, \dots, n_k; \lambda) = f(n_1; \lambda)f(n_2; \lambda) \dots f(n_k; \lambda) = \prod_{i=1}^k f(n_i; \lambda) \quad (6.18)$$

The  $n_1, n_2, \dots, n_k$  are observed (known values) and  $\lambda$  is unknown. The function  $L(\lambda)$  is called the likelihood function. To estimate the value of  $\lambda$  by the method of maximum likelihood, we must find the value of  $\lambda$  that maximizes the function  $L(\lambda)$ . The interpretation is that we are finding the value of the parameter that is maximizing the likelihood that it came from this sample.

Substituting the definition of the Poisson distribution into the likelihood function yields

$$\begin{aligned} L(\lambda) &= \prod_{i=1}^k \frac{e^{-\lambda}(\lambda)^{n_i}}{n_i!} \\ &= \frac{e^{-k\lambda} \lambda^{\sum_{i=1}^k n_i}}{\prod_{i=1}^k n_i!} \end{aligned}$$

It can be shown that maximizing  $L(\lambda)$  is the same as maximizing  $\ln(L(\lambda))$ . This is called the log-likelihood function. Thus,

$$\ln(L(\lambda)) = -k\lambda + \ln(\lambda) \sum_{i=1}^k n_i - \sum_{i=1}^k \ln(n_i!)$$
(6.19)

Differentiating with respect to  $\lambda$  yields

$$\frac{d\ln(L(\lambda))}{d\lambda} = -k + \frac{\sum_{i=1}^k n_i}{\lambda}$$
(6.20)

When we set this equal to zero and solve for  $\lambda$ , we get

$$0 = -k + \frac{\sum_{i=1}^k n_i}{\lambda}$$
(6.21)

$$\hat{\lambda} = \frac{\sum_{i=1}^k n_i}{k}$$
(6.22)

If the second derivative  $\frac{d^2 \ln L(\lambda)}{d\lambda^2} < 0$ , then a maximum is obtained.

$$\frac{d^2 \ln L(\lambda)}{d\lambda^2} = \frac{-\sum_{i=1}^k n_i}{\lambda^2}$$
(6.23)

Because the  $n_i$  are positive and  $\lambda$  is positive the second derivative must be negative; therefore, the maximum likelihood estimator for the parameter of the Poisson distribution is given by Equation 6.22. Notice that this is the sample average of the interval counts.

While the Poisson distribution has an analytical form for the maximum likelihood estimator, not all distributions will have this property. Estimating the parameters of distributions will, in general, involve nonlinear optimization. This motivates the use of software tools such as R when performing the analysis. Software tools will perform this estimation process with little difficulty. Let us complete this example using R to fit and test whether or not the Poisson distribution is a good model for this data.

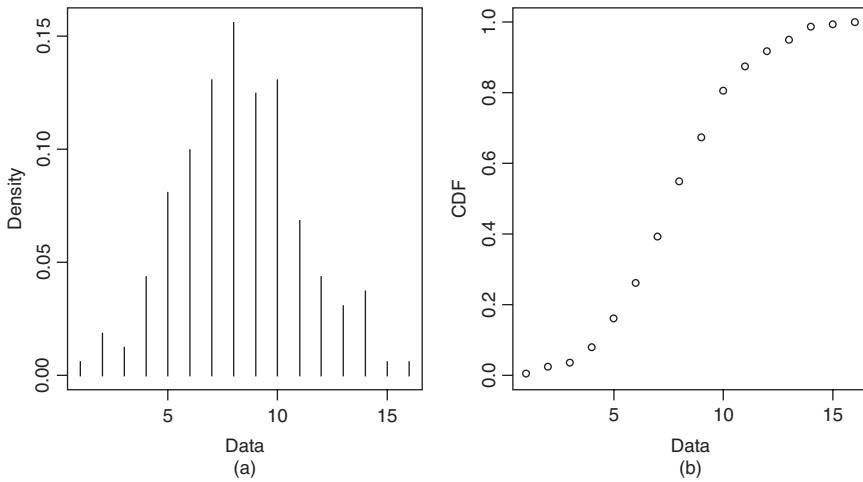
#### 6.5.4 Checking the Goodness of Fit of the Model

Fortunately, R has a very useful package for fitting a wide variety of discrete and continuous distributions call the *fitdistrplus* package. To install and load the package, do the following:

```
> install.packages('fitdistrplus')
> library(fitdistrplus)
> plotdist(p2$N, discrete = TRUE)
```

The *plotdist* command will plot the empirical PMF and the CDF as illustrated in Figure 6.8.

To perform the fitting and analysis, you use the *fitdist* and *gofstat* commands. In addition, plotting the output from the *fitdist* function will provide a comparison of the empirical distribution to the theoretical distribution (Figure 6.9).

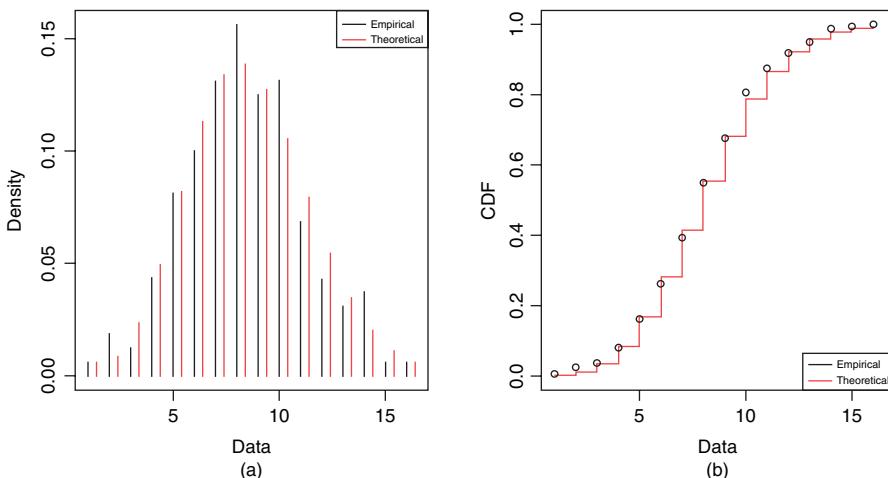


**Figure 6.8** Plot of the empirical (a) PMF and (b) CDF of the computer laboratory arrival counts.

```
> fp = fitdist(p2$N, "pois")
> summary(fp)
Fitting of the distribution ' pois ' by maximum likelihood
Parameters :
      estimate Std. Error
lambda     8.275  0.2274176
Loglikelihood: -393.7743   AIC: 789.5485   BIC: 792.6237
```

The output of the *fitdist* and the *summary* commands provides the estimate of  $\lambda = 8.275$ . The result object, *fp*, returned by the *fitdist* can then subsequently be used to plot the fit, *plot*, and perform a goodness-of-fit test, *gofstat*.

```
> plot(fp)
```



**Figure 6.9** Plot of the empirical and theoretical (a) PMF and (b) CDF of the computer laboratory arrival counts.

```

> gofstat(fp)
Chi-squared statistic: 2.233903
Degree of freedom of the Chi-squared distribution: 7
Chi-squared p-value: 0.9457686
Chi-squared table:
  obscounts theocounts
<= 4    13.00000 13.58936
<= 5    13.00000 13.18232
<= 6    16.00000 18.18062
<= 7    21.00000 21.49209
<= 8    25.00000 22.23088
<= 9    20.00000 20.44006
<= 10   21.00000 16.91415
<= 12   18.00000 21.49835
> 12    13.00000 12.47218
Goodness-of-fit criteria
                           1-mle-pois
Aikake's Information Criterion    789.5485
Bayesian Information Criterion    792.6237

```

The *gofstat* command performs a chi-squared goodness-of-fit test and computes the chi-squared statistic value (here 2.233903) and the *p*-value (0.9457686). Clearly, the chi-squared test statistic *p*-value is quite high. Again the null hypothesis is that the observations come from a Poisson distribution with the alternative that they do not. The high *p*-value suggests that we should not reject the null hypothesis and conclude that a Poisson distribution is a very reasonable model for the computer laboratory arrival counts.

## EXAMPLE 6.2 Fitting a Discrete Empirical Distribution

We are interested in modeling the number of packages delivered on a small parcel truck to a hospital loading dock. A distribution for this random variable will be used in a loading dock simulation to understand the ability of the workers to unload the truck in a timely manner. Unfortunately, a limited sample of observations is available from 40 different truck shipments. The following table provides the data.

130	130	110	130	130	110	110	130	120	130
140	140	130	110	110	140	130	140	130	120
140	150	120	150	120	130	120	100	110	150
130	120	120	130	120	120	130	130	130	100

Develop a probability distribution model for the number of packages per truck shipment.

Developing a probability model will be a challenge for this data set because of the limited amount of data. Using the following *R* commands, we can get a good understanding of the data. Assuming that the data has been read into the variable, *packageCnt*, the *hist()*, *stripchart()*, and *table()* commands provide an initial analysis.

```

> hist(packageCnt, main="Packages per Shipment", xlab="#packages")
> stripchart(packageCnt, method="stack", pch="o")
> table(packageCnt)

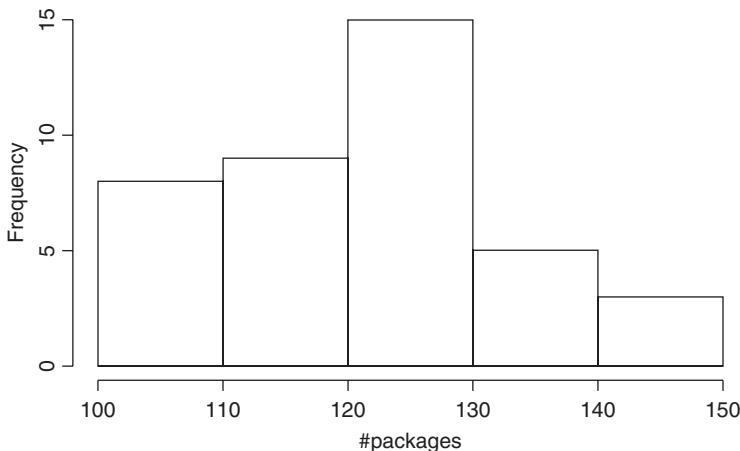
```

```

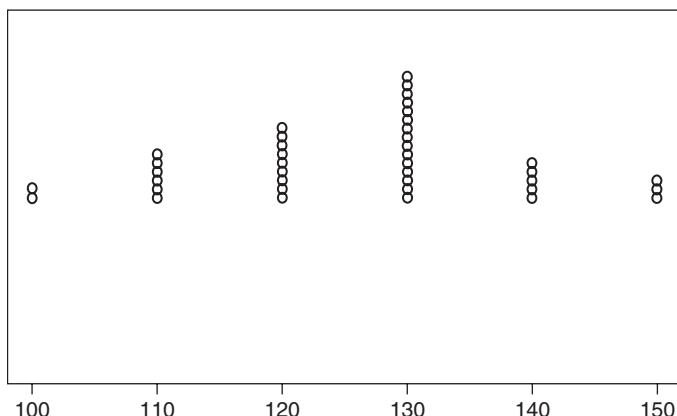
packageCnt
100 110 120 130 140 150
 2   6   9  15   5   3
> summary(packageCnt)
  Min. 1st Qu. Median      Mean 3rd Qu.      Max.
 100      120      130      126      130      150
> tp = table(packageCnt)
> tp/40
 100     110     120     130     140     150
0.050  0.150  0.225  0.375  0.125  0.075

```

As we can see from the *R* output, the range of the data varies between 100 and 150. The histogram, shown in Figure 6.10, illustrates the shape of the data. It appears to be slightly positively skewed. Figure 6.11 presents a dot plot of the observed counts. From this figure, we can clearly see that the only values obtained in the sample are 100, 110, 120, 130, 140, and 150. It looks as though the ordering comes in tens, starting at 100 units.



**Figure 6.10** Histogram of package count per shipment.



**Figure 6.11** Dot plot of package count per shipment.

Because of the limited size of the sample and limited variety of data points, fitting a distribution will be problematic. However, we can fit a discrete empirical distribution to the proportions observed in the sample. Using the `table()` command, we can summarize the counts. Then, by dividing by 40 (the total number of observations), we can get the proportions as shown in the R listing. Thus, we can represent this situation with the following probability mass and CDFs given in Figure 6.12. Therefore, we can represent the number of packages in a shipment using the `DISC()` function as `DISC(0.05, 100, 0.2, 110, 0.425, 120, 0.80, 130, 0.925, 140, 1.0, 150)`.

$$P\{X = x\} = \begin{cases} 0.05 & x = 100 \\ 0.15 & x = 110 \\ 0.225 & x = 120 \\ 0.375 & x = 130 \\ 0.125 & x = 140 \\ 0.075 & x = 150 \end{cases} \quad F(x) = \begin{cases} 0.0 & \text{if } x < 100 \\ 0.05 & \text{if } 100 \leq x < 110 \\ 0.20 & \text{if } 110 \leq x < 120 \\ 0.425 & \text{if } 120 \leq x < 130 \\ 0.80 & \text{if } 130 \leq x < 140 \\ 0.925 & \text{if } 140 \leq x < 150 \\ 1.0 & \text{if } x \geq 150 \end{cases}$$

**Figure 6.12** PMF and CDF for package count per shipment.

## 6.6 FITTING CONTINUOUS DISTRIBUTIONS

In the previous section, we examined the modeling of a discrete distribution. In this section, we look at modeling a continuous distribution using the functionality available in R. This example starts with step 3 of the input modeling process. That is, the data has already been collected. Additional discussion of this topic can be found in Chapter 6 of Law (2007).

### ■ EXAMPLE 6.3 Fitting a Continuous Distribution to Task Times

Suppose that we are interested in modeling the time that it takes to perform a computer component repair task. The 100 observations are provided below in minutes. Fit an appropriate distribution to this data.

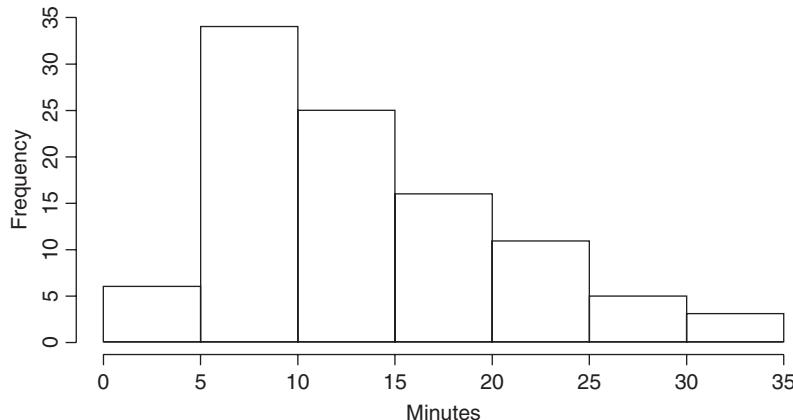
	1	2	3	4	5	6	7	8	9	10
1	15.3	10.0	12.6	19.7	9.4	11.7	22.6	13.8	15.8	17.2
2	12.4	3.0	6.3	7.8	1.3	8.9	10.2	5.4	5.7	28.9
3	16.5	15.6	13.4	12.0	8.2	12.4	6.6	19.7	13.7	17.2
4	3.8	9.1	27.0	9.7	2.3	9.6	8.3	8.6	14.8	11.1
5	19.5	5.3	25.1	13.5	24.7	9.7	21.0	3.9	6.2	10.9
6	7.0	10.5	16.1	5.2	23.0	16.0	11.3	7.2	8.9	7.8
7	20.1	17.8	14.4	8.4	12.1	3.6	10.9	19.6	14.1	16.1
8	11.8	9.2	31.4	16.4	5.1	20.7	14.7	22.5	22.1	22.7
9	22.8	17.7	25.6	10.1	8.2	24.4	30.8	8.9	8.1	12.9
10	9.8	5.5	7.4	31.5	29.1	8.9	10.3	8.0	10.9	6.2

### 6.6.1 Visualizing the Data

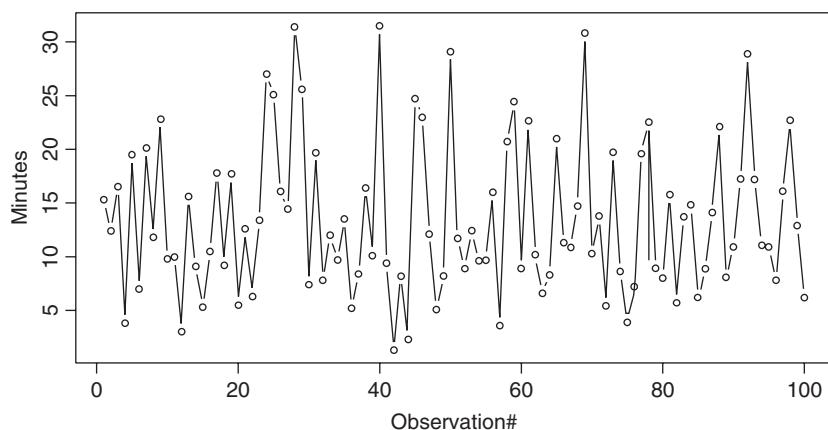
The first steps are to visualize the data and check for independence. This can be readily accomplished using the *hist*, *plot*, and *acf* functions in R. Assume that the data is in a file called, *taskTimes.txt* within the R working directory.

```
> y = scan(file="taskTimes.txt")
> hist(y, main="Task Times", xlab = "minutes")
> plot(y,type="b",main="Task Times", ylab = "minutes",
>       xlab = "Observation#")
> acf(y, main = "ACF Plot for Task Times")
```

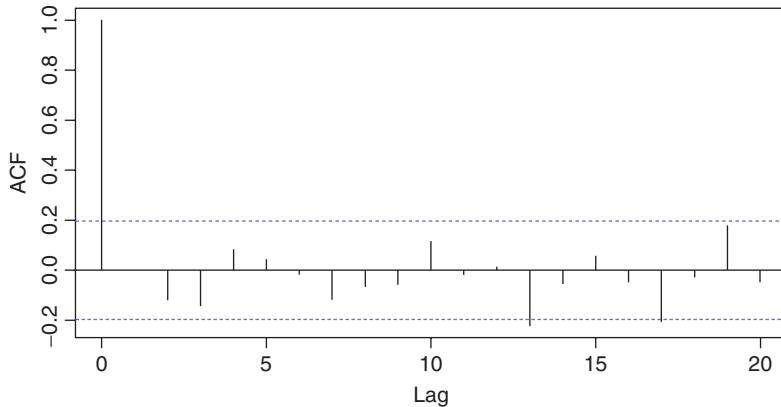
As can be seen in Figure 6.13, the histogram is slightly right skewed. The time series plot, Figure 6.14, illustrates no significant pattern (e.g., trends). Finally, the autocorrelation plot, Figure 6.15, shows no significant correlation (all lags are well within the confidence band) with respect to observation number.



**Figure 6.13** Histogram of computer repair task times.



**Figure 6.14** Time series plot of computer repair task times.



**Figure 6.15** ACF plot for computer repair task times.

Based on the visual analysis, we can conclude that the task times are likely to be independent and identically distributed.

### 6.6.2 Statistically Summarize the Data

An analysis of the statistical properties of the task times can be easily accomplished in R using the *summary*, *mean*, *var*, *sd*, and *t.test* functions.

```
> summary(y)
   Min. 1st Qu. Median     Mean 3rd Qu.    Max.
1.300  8.275 11.750 13.410 17.320 31.500
> mean(y)
[1] 13.412
> var(y)
[1] 50.44895
> sd(y)
[1] 7.102742
> t.test(y)
One Sample t-test
data: y
t = 18.8828, df = 99, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval: 12.00266 14.82134
sample estimates: mean of x 13.412
```

The *summary* command summarizes the distributional properties in terms of the minimum, maximum, median, and first and third quartiles of the data. The *mean*, *var*, and *sd* commands compute the sample average, sample variance, and sample standard deviation of the data. Finally, the *t.test* command can be used to form a 95% confidence interval on the mean and test if the true mean is significantly different from zero. The *descdist* command of the *fitdistrplus* package will also provide a description of the distribution's properties.

```
> descdist(y)
summary statistics
```

---

```

min: 1.3   max: 31.5
median: 11.75
mean: 13.412
estimated sd: 7.102742
estimated skewness: 0.7433715
estimated kurtosis: 2.905865

```

The median is less than the mean and the skewness is less than 1.0. This confirms the visual conclusion that the data is slightly skewed to the right. Before continuing with the analysis, let us recap what has been learned so far.

- The data appears to be stationary. This conclusion is based on the time series plot where no discernible trend with respect to time is found in the data.
- The data appears to be independent. This conclusion is from the autocorrelation plot.
- The distribution of the data is positively (right) skewed and unimodal. This conclusion is based on the histogram and from the statistical summary.

### 6.6.3 Hypothesizing and Testing a Distribution

The next steps involve the model fitting processes of hypothesizing distributions, estimating the parameters, and checking for goodness of fit.

Distributions such as the gamma, Weibull, and lognormal should be candidates for this situation based on the histogram. We will perform the analysis for the gamma distribution “by hand” so that you can develop an understanding of the process. Then, the *fitdistrplus* package will be illustrated.

The following is what we are going to do.

1. Perform a chi-squared goodness-of-fit test.
2. Perform a K-S goodness-of-fit test.
3. Examine the P-P and Q-Q plots.

Recall that the chi-square test divides the range of the data,  $x_1, x_2, \dots, x_n$ , into,  $k$ , intervals and tests if the number of observations that fall in each interval is close to the expected number that should fall in the interval, given the hypothesized distribution is the correct model.

Since a histogram tabulates the necessary counts for the intervals, it is useful to begin the analysis by developing a histogram. Let  $b_0, b_1, \dots, b_k$  be the breakpoints (end points) of the class intervals such that  $(b_0, b_1], (b_1, b_2], \dots, (b_{k-1}, b_k]$  form  $k$  disjoint and adjacent intervals. The intervals do not have to be of equal width. Also,  $b_0$  can be equal to  $-\infty$  resulting in interval  $(-\infty, b_1]$  and  $b_k$  can be equal to  $+\infty$  resulting in interval  $(b_{k-1}, +\infty)$ . Define  $\Delta b_j = b_j - b_{j-1}$  and if all the intervals have the same width (except perhaps for the end intervals),  $\Delta b = \Delta b_j$ .

To count the number of observations that fall in each interval, recall the following function from Chapter 2:

$$c(\vec{x} \leq b) = \#\{x_i \leq b\}, i = 1, \dots, n$$

$c(\vec{x} \leq b)$  counts the number of observations less than or equal to  $x$ . Let  $c_j$  be the observed count of the  $x$  values contained in the  $j$ th interval  $(b_{j-1}, b_j]$ . Then, we can determine  $c_j$  via the following equation:

$$c_j = c(\vec{x} \leq b_j) - c(\vec{x} \leq b_{j-1})$$

Define  $h_j = c_j/n$  as the relative frequency for the  $j$ th interval. Note that  $\sum_{j=1}^k h_j = 1$ . A plot of the cumulative relative frequency,  $\sum_{i=1}^j h_i$ , for each  $j$  is called a cumulative distribution plot. A plot of  $h_j$  should resemble the true probability distribution in shape because according to the mean value theorem of calculus.

$$p_j = P\{b_{j-1} \leq X \leq b_j\} = \int_{b_{j-1}}^{b_j} f(x) dx = \Delta b \times f(y) \text{ for } y \in (b_{j-1}, b_j) \quad (6.24)$$

Therefore, since  $h_j$  is an estimate for  $p_j$ , the shape of the distribution should be proportional to the relative frequency, that is,  $h_j \approx \Delta b \times f(y)$ .

The number of intervals is a key decision parameter and will affect the visual quality of the histogram and ultimately the chi-squared test statistic calculations that are based on the tabulated counts from the histogram. In general, the visual display of the histogram is highly dependent upon the number of class intervals. If the widths of the intervals are too small, the histogram will tend to have a ragged shape. If the widths of the intervals are too large, the resulting histogram will be very block like. Two common rules for setting the number of interval are

1. Square root rule, choose the number of intervals,  $k = \sqrt{n}$ .
2. Sturges' rule, choose the number of intervals,  $k = \lceil 1 + \log_2(n) \rceil$ .

A frequency diagram in R is very simple by using the `hist()` function. The `hist()` function provides the frequency version of histogram and `hist(x, freq=F)` provides the density version of the histogram. The `hist()` function will automatically determine breakpoints using the Sturges rule as its default. You can also provide your own breakpoints in a vector. The `hist()` function will automatically compute the counts associated with the intervals. The commands for doing this are provided in the following listing.

```
> h = hist(y)
> h
$breaks
[1] 0 5 10 15 20 25 30 35
$counts
[1] 6 34 25 16 11 5 3
$intensities
[1] 0.012 0.068 0.050 0.032 0.022 0.010 0.006
$density
[1] 0.012 0.068 0.050 0.032 0.022 0.010 0.006
$mids
[1] 2.5 7.5 12.5 17.5 22.5 27.5 32.5
$xname
[1] "y"
$equidist
[1] TRUE
```

```

attr(", "class")
[1] "histogram"
> b = c(0,4,8,12,16,20,24,28,32)
> hb = hist(y, breaks = b)
> hb
$breaks
[1] 0 4 8 12 16 20 24 28 32
$counts
[1] 6 16 30 17 12 9 5 5
etc.

```

Notice how the *hist* command returns a result object. In the example, the result object is assigned to the variable *h*. By printing the result object, you can see all the tabulated results. For example, the variable *h\$counts* shows the tabulation of the counts based on the default breakpoints. The breakpoints are given by the variable *h\$breaks*. Note that by default, *hist* defines the intervals as right closed, that is,  $(b_{k-1}, b_k]$ , rather than left closed,  $[b_{k-1}, b_k)$ . If you want left-closed intervals, set the *hist* parameter, *right = FALSE*. The relative frequencies,  $h_j$ , can be computed by dividing the counts by the number of observations, that is, *h\$counts/length(y)*.

The variable *h\$density* holds the relative frequencies divided by the interval length. In terms of notation, this is  $f_j = h_j / \Delta b_j$ . This is referred to as the density because it estimates the height of the probability density curve.

To define your own break points, put them in a vector using the *vector* command (e.g., *b = c(0,4,8,12,16,20,24,28,32)*) and then specify the vector with the *breaks* option of the *hist* command if you do not want to use the default breakpoints. The following listing illustrates how to do this.

```

> b = c(0,4,8,12,16,20,24,28,32)
> hb = hist(y, breaks = b)
> hb
$breaks
[1] 0 4 8 12 16 20 24 28 32
$counts
[1] 6 16 30 17 12 9 5 5
etc.

```

You can also use the *cut()* function and the *table()* command to tabulate the counts by providing a vector of breaks and tabulate the counts using the *cut()* and the *table()* commands without using the *hist* command. The following listing illustrates how to do this.

```

> b = c(0, 4, 8, 12, 16, 20, 24, 28, 32) #define the breaks in a vector
> y.cut = cut(y, breaks=b) #define the intervals
> table(y.cut)
y.cut
(0,4]   (4,8]   (8,12]  (12,16] (16,20] (20,24] (24,28] (28,32]
       6        16       30       17       12        9        5        5

```

Thus, using the *hist* function in R, we have a method for tabulating the relative frequencies. In order to apply the chi-square test, we need to be able to compute the following test

statistic:

$$\chi^2_0 = \sum_{j=1}^k \frac{(c_j - np_j)^2}{np_j} \quad (6.25)$$

where

$$p_j = P\{b_{j-1} \leq X \leq b_j\} = \int_{b_{j-1}}^{b_j} f(x) dx = F(b_j) - F(b_{j-1}) \quad (6.26)$$

Notice that  $p_j$  depends on  $F(x)$ , the CDF of the hypothesized distribution. Thus, we need to hypothesize a distribution and estimate the parameters of the distribution.

For this situation, we will hypothesize that the task times come from a gamma distribution. Therefore, we need to estimate the shape ( $\alpha$ ) and the scale ( $\beta$ ) parameters. In order to do this, we can use an estimation technique such as the method of moments or the maximum likelihood method. For simplicity and illustrative purposes, we will use the method of moments to estimate the parameters.

The method of moments is a technique for constructing estimators of the parameters that is based on matching the sample moments (e.g., sample average and sample variance) with the corresponding distribution moments. This method equates sample moments to population (theoretical) ones. Recall that the mean and variance of the gamma distribution are

$$E [X] = \alpha\beta \quad (6.27)$$

$$\text{Var} [X] = \alpha\beta^2 \quad (6.28)$$

Setting  $\bar{X} = E [X]$  and  $S^2 = \text{Var} [X]$  and solving for  $\alpha$  and  $\beta$  yields

$$\hat{\alpha} = \frac{(\bar{X})^2}{S^2} \quad (6.29)$$

$$\hat{\beta} = \frac{S^2}{\bar{X}} \quad (6.30)$$

Using the results,  $\bar{X} = 13.412$  and  $S^2 = 50.44895$ , yields

$$\hat{\alpha} = \frac{(\bar{X})^2}{S^2} = \frac{(13.412)^2}{50.44895} = 3.56562$$

$$\hat{\beta} = \frac{S^2}{\bar{X}} = \frac{50.44895}{13.412} = 3.761478$$

Then, you can compute the theoretical probability of falling in your intervals. Table 6.4 illustrates the computations necessary to compute the chi-squared test statistic.

Since the seventh and eighth intervals have less than 5 expected counts, it is recommended to combine them with the sixth interval. Computing the chi-square test statistic

**TABLE 6.4 Chi-Squared Goodness-of-Fit Calculations**

$j$	$b_{j-1}$	$b_j$	$c_j$	$F(b_{j-1})$	$F(b_j)$	$p_j$	$np_j$	$\frac{(c_j - np_j)^2}{np_j}$
1	0.00	5.00	6.00	0.00	0.08	0.08	7.89	0.45
2	5.00	10.00	34.00	0.08	0.36	0.29	28.54	1.05
3	10.00	15.00	25.00	0.36	0.65	0.29	28.79	0.50
4	15.00	20.00	16.00	0.65	0.84	0.18	18.42	0.32
5	20.00	25.00	11.00	0.84	0.93	0.09	9.41	0.27
6	25.00	30.00	5.00	0.93	0.97	0.04	4.20	0.15
7	30.00	35.00	3.00	0.97	0.99	0.02	1.72	0.96
8	35.00	$\infty$	0.00	0.99	1.00	0.01	1.03	1.03

value over the six intervals yields

$$\begin{aligned}\chi_0^2 &= \sum_{j=1}^6 \frac{(c_j - np_j)^2}{np_j} \\ &= \frac{(6.0 - 7.89)^2}{7.89} + \frac{(34 - 28.54)^2}{28.54} + \frac{(25 - 28.79)^2}{28.79} + \frac{(16 - 18.42)^2}{218.42} \\ &\quad + \frac{(11 - 9.41)^2}{9.41} + \frac{(8 - 6.95)^2}{6.95} \\ &= 2.74\end{aligned}$$

Since two parameters of the gamma were estimated from the data, the degrees of freedom for the chi-square test is  $3$  (#intervals - #parameters - 1 = 6-2-1). Computing the  $p$ -value yields  $P\{\chi_3^2 > 2.74\} = 0.433$ . Thus, given such a high  $p$ -value, we would not reject the hypothesis that the observed data is gamma distributed with  $\alpha = 3.56562$  and  $\beta = 3.761478$ .

The following listing provides a script that will compute the chi-square test statistic and its  $p$ -value within R.

```
y = scan(file="taskTimes.txt") #read in the file
a = mean(y)*mean(y)/var(y) #estimate alpha
b = var(y)/mean(y) #estmate beta
hy = hist(y, main="Task Times", xlab = "minutes") # make histogram
LL = hy$breaks # set lower limit of intervals
UL = c(LL[-1],10000) # set upper limit of intervals
FLL = pgamma(LL,shape = a, scale = b) #compute F(LL)
FUL = pgamma(UL,shape = a, scale = b) #compute F(UL)
pj = FUL - FLL # compute prob of being in interval
ej = length(y)*pj # compute expected number in interval
e = c(ej[1:5],sum(ej[6:8])) #combine last 3
intervals
cnts = c(hycounts[1:5],sum(hycounts[6:7])) #combine last 3 inter-
vals
chissq = ((cnts-e)^{2})/e #compute chi sq values
sumchisq = sum(chissq) # compute test statistic
```

```

df = length(e)-2-1 #compute degrees of freedom
pvalue = 1 - pchisq(sumchisq, df) #compute p-value
print(sumchisq) # print test statistic
print(pvalue) #print p-value

```

Recall that the K-S test statistic,  $D_n$ , is defined as  $D_n = \max\{D_n^+, D_n^-\}$  where

$$\begin{aligned}
D_n^+ &= \max_{1 \leq i \leq n} \{\tilde{F}_n(x_{(i)}) - \hat{F}(x_{(i)})\} \\
&= \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - \hat{F}(x_{(i)}) \right\} \\
D_n^- &= \max_{1 \leq i \leq n} \{\hat{F}(x_{(i)}) - \tilde{F}_n(x_{(i-1)})\} \\
&= \max_{1 \leq i \leq n} \left\{ \hat{F}(x_{(i)}) - \frac{i-1}{n} \right\}
\end{aligned}$$

where  $x_{(i)}$  is the  $i$ th order statistic and

$$\begin{aligned}
\tilde{F}_n(x) &= \frac{c(\vec{x} \leq x)}{n} \\
\tilde{F}_n(x_{(i)}) &= \frac{i}{n}
\end{aligned}$$

The following R listing illustrates how to compute the K-S statistic by hand (which is quite unnecessary) because you can simply use the *ks.test* command as illustrated.

```

> j = 1:length(y) # make a vector to count y's
> yj = sort(y) # sort the y's
> Fj = pgamma(yj, shape = a, scale = b) #compute F(yj)
> n = length(y)
> D = max(max((j/n)-Fj), max(Fj - ((j-1)/n))) # compute K-S test
statistic
> print(D)
[1] 0.05265431
> ks.test(y, 'pgamma', shape=a, scale=b) # compute k-s test
One-sample Kolmogorov-Smirnov test
data: y
D = 0.0527, p-value = 0.9444
alternative hypothesis: two-sided

```

Based on the very high  $p$ -value of 0.9444, we should not reject the hypothesis that the observed data is gamma distributed with  $\alpha = 3.56562$  and  $\beta = 3.761478$ .

We have now completed the chi-squared goodness-of-fit test as well as the K-S test. The chi-squared test has more general applicability than the K-S test. Specifically, the chi-squared test applies to both continuous and discrete data; however, it suffers from depending on the interval specification. In addition, it has a number of other shortcomings which are discussed in Law (2007). While the K-S test can also be applied to discrete data, special tables must be used for getting the critical values. Additionally, the K-S test in its original form assumes that the parameters of the hypothesized distribution are known, that is, given without estimating from the data. Research on the effect of using the K-S test with estimated parameters has indicated that it will be conservative in the sense that the

actual Type I error will be less than specified. Additional advantage and disadvantage of the K-S test are given in Law (2007). There are other statistical tests that have been devised for testing the goodness of fit for distributions. One such test is Anderson–Darling test. Law (2007) described this test. This test detects tail differences and has a higher power than the K-S test for many popular distributions. It can be found as standard output in commercial distribution fitting software.

#### 6.6.4 Visualizing the Fit

Another valuable diagnostic tool is to make probability-probability (P-P) plots and quantile-quantile (Q-Q) plots.

A P-P plot plots the empirical distribution function versus the theoretical distribution evaluated at each order statistic value. Recall that the empirical distribution is defined as

$$\tilde{F}_n(x_{(i)}) = \frac{i}{n}$$

Alternative definitions are also used in many software packages to account for continuous data. As previously mentioned,

$$\tilde{F}_n(x_{(i)}) = \frac{i - 0.5}{n}$$

is very common as well as

$$\tilde{F}_n(x_{(i)}) = \frac{i - 0.375}{n + 0.25}$$

To make a P-P plot, perform the following steps:

1. Sort the data to obtain the order statistics:  $(x_{(1)}, x_{(2)}, \dots, x_{(n)})$
2. Compute  $\tilde{F}_n(x_{(i)}) = \frac{i-0.5}{n} = q_i$  for  $i = 1, 2, \dots, n$
3. Compute  $\hat{F}(x_{(i)})$  for  $i = 1, 2, \dots, n$  where  $\hat{F}$  is the CDF of the hypothesized distribution
4. Plot  $\hat{F}(x_{(i)})$  versus  $\tilde{F}_n(x_{(i)})$  for  $i = 1, 2, \dots, n$ .

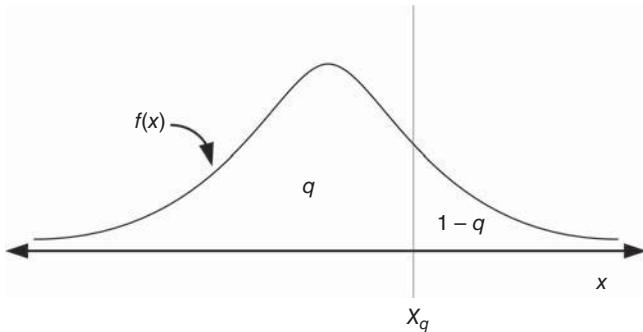
The Q-Q plot is similar in spirit to the P-P plot. For the Q-Q plot, the quantiles of the empirical distribution (which are simply the order statistics) are plotted versus the quantiles from the hypothesized distribution. Let  $0 \leq q \leq 1$  so that the  $q$ th quantile of the distribution is denoted as  $x_q$  and is defined by

$$q = P(X \leq x_q) = F(x_q) = \int_{-\infty}^{x_q} f(u) du \quad (6.31)$$

Thus,  $x_q$  is that value on the measurement axis such that  $100q\%$  of the area under the graph of  $f(x)$  lies to the left of  $x_q$  and  $100(1 - q)\%$  of the area lies to the right. This is the same as the inverse CDF as described in Chapter 2.

For example, the  $z$ -values for the standard normal distribution tables are the quantiles of that distribution. The quantiles of a distribution are readily available if the inverse CDF of the distribution is available (Figure 6.16). Thus, the quantile can be defined as

$$x_q = F^{-1}(q)$$



**Figure 6.16** The quantile of a distribution.

where  $F^{-1}$  represents the inverse of the CDF (not the reciprocal). For example, if the hypothesized distribution is  $N(0,1)$ , then  $1.96 = \Phi^{-1}(0.975)$  so that  $x_{0.975} = 1.96$  where  $\Phi(z)$  is the CDF of the standard normal distribution. When you give a probability to the inverse of the CDF, you get back the corresponding ordinate value that is associated with the area under the curve, for example, the quantile.

To make a Q–Q plot, perform the following steps:

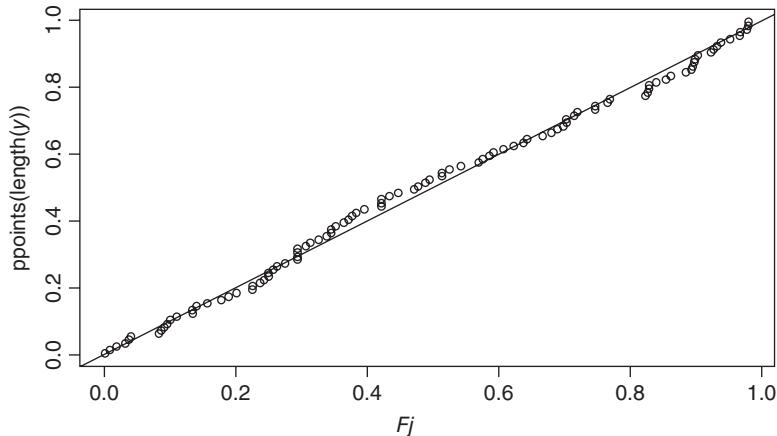
1. Sort the data to obtain the order statistics:  $(x_{(1)}, x_{(2)}, \dots, x_{(n)})$
2. Compute  $q_i = \frac{i-0.5}{n}$  for  $i = 1, 2, \dots, n$
3. Compute  $x_{q_i} = \hat{F}^{-1}(q_i)$  for where  $i = 1, 2, \dots, n$  is the  $\hat{F}^{-1}$  inverse CDF of the hypothesized distribution
4. Plot  $x_{q_i}$  versus  $x_{(i)}$  for  $i = 1, 2, \dots, n$ .

Thus, in order to make a P–P plot, the CDF of the hypothesized distribution must be available, and in order to make a Q–Q plot, the inverse CDF of the hypothesized distribution must be available. When the inverse CDF is not readily available, there are other methods to making Q–Q plots for many distributions. These methods are outlined in Law (2007). The following example will illustrate how to make and interpret the P–P plot and Q–Q plot for the hypothesized gamma distribution for the task times.

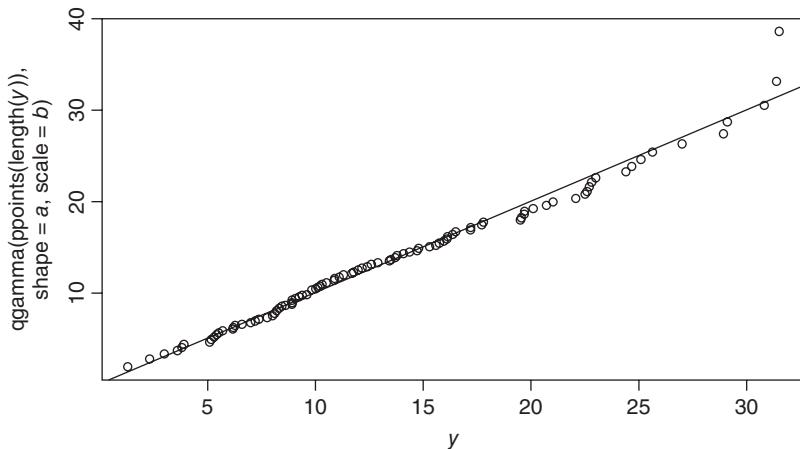
The following R listing will make the P–P and the Q–Q plots for this situation.

```
> plot(Fj, ppoints(length(y))) # make P-P plot
> abline(0,1) # add a reference line to the plot
> qqplot(y, qgamma(ppoints(length(y)), shape = a, scale = b)) # 
make Q-Q Plot
> abline(0,1) # add a reference line to the plot
```

The function `ppoints()` in R will generate  $\tilde{F}_n(x_{(i)})$ . Then you can easily use the distribution function (with the “p,” as in `pgamma()`) to compute the theoretical probabilities. In R, the quantile function can be found by appending a “q” to the name of the available distributions. We have already seen `qt()` for the student-*t* distribution. For the normal, we use `qnorm()` and for the gamma, we use `qgamma()`. Search the R help for “distributions” to find the other common distributions. The function `abline()` will add a reference line between 0 and 1 to the plot.



**Figure 6.17** The P-P plot for the task times with gamma( $\alpha = 3.56, \beta = 3.76$ ).



**Figure 6.18** The Q-Q plot for the task times with gamma( $\alpha = 3.56, \beta = 3.76$ ).

The Q-Q plot should appear approximately linear with intercept 0 and slope 1, that is, a  $45^\circ$  line, if there is a good fit to the data. In addition, curvature at the ends implies too long or too short tails, convex or concave curvature implies asymmetry, and stragglers at either ends may be outliers. The P-P plot should also appear linear with intercept 0 and slope 1. The `abline()` function was used to add the reference line to the plots. Figure 6.17 illustrates the P-P plot and Figure 6.18 illustrates the Q-Q plot. As can be seen in the figures, both plots do not appear to show any significant departure from a straight line. Notice that the Q-Q plot is a little off in the right tail.

Now, that we have seen how to do the analysis “by hand,” let us see how easy it can be using the *fitdistrplus* package. Notice that the `fitdist` command will fit the parameters of the distribution. Then, the `gofstat` function does all the work to compute the chi-square goodness of fit, K-S test statistic, as well as other goodness-of-fit criteria. The results lead to the same conclusion that we had before: the gamma distribution is a good model for this data.

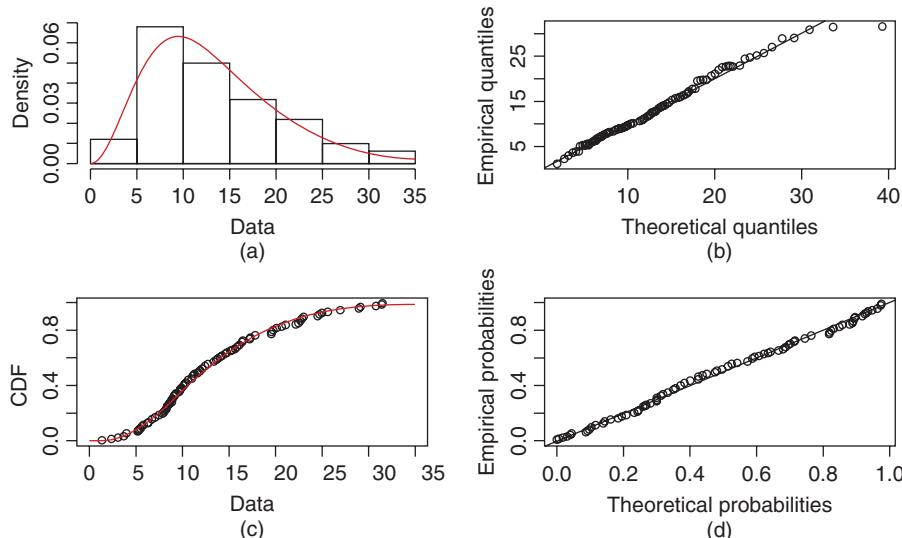
```

> library(fitdistrplus)
> fy = fitdist(y, "gamma")
> print(fy)
Fitting of the distribution ' gamma ' by maximum likelihood
Parameters:
      estimate Std. Error
shape 3.4098479 0.46055722
rate  0.2542252 0.03699365
> plot(fy)
> gfy = gofstat(fy)
> print(gfy)
Goodness-of-fit statistics
                               1-mle-gamma
Kolmogorov-Smirnov statistic 0.04930008
Cramer-von Mises statistic  0.03754480
Anderson-Darling statistic   0.25485917

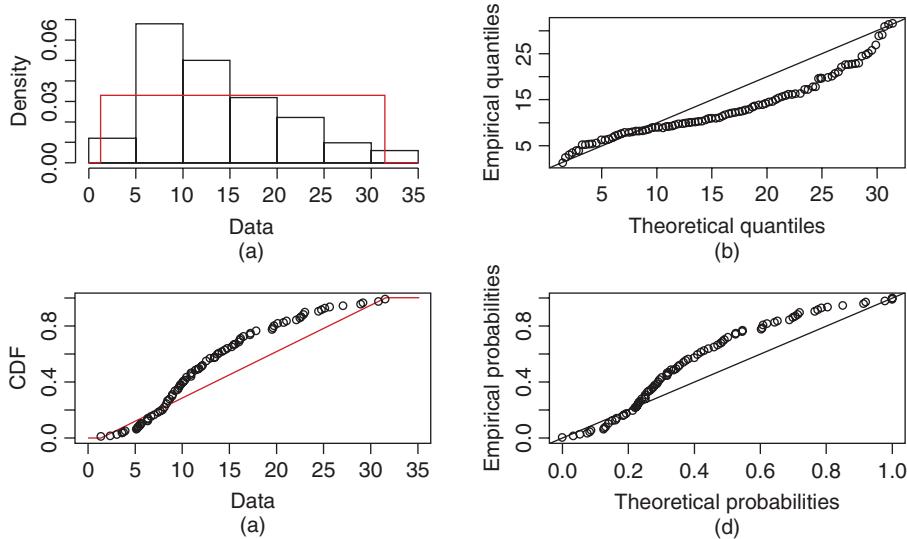
Goodness-of-fit criteria
                               1-mle-gamma
Aikake's Information Criterion     663.3157
Bayesian Information Criterion    668.5260
> print(gfy$chisq)
[1] 3.544766
> print(gfy$chisq$pvalue)
[1] 0.8956877
> print(gfy$chisqdf)
[1] 8

```

Plotting the object returned from the *fitdist* command produces a plot (Figure 6.19) of the empirical and theoretical distributions, as well as the P–P and Q–Q plots. Figure 6.20



**Figure 6.19** Fit distribution plot from *fitdistrplus* for gamma distribution fit. (a) Empirical and theoretical density, (b) Q–Q plot, (c) empirical and theoretical CDFs, and (d) P–P plot.



**Figure 6.20** Fit distribution plot from *fitdistrplus* for uniform distribution fit. (a) Empirical and theoretical density, (b) Q–Q plot, (c) empirical and theoretical CDFs, and (d) P–P plot.

illustrates the P–P and Q–Q plots if we were to hypothesize a uniform distribution. Clearly, the plots in Figure 6.20 illustrate that a uniform distribution is not a good model for the task times.

## 6.7 USING THE INPUT ANALYZER

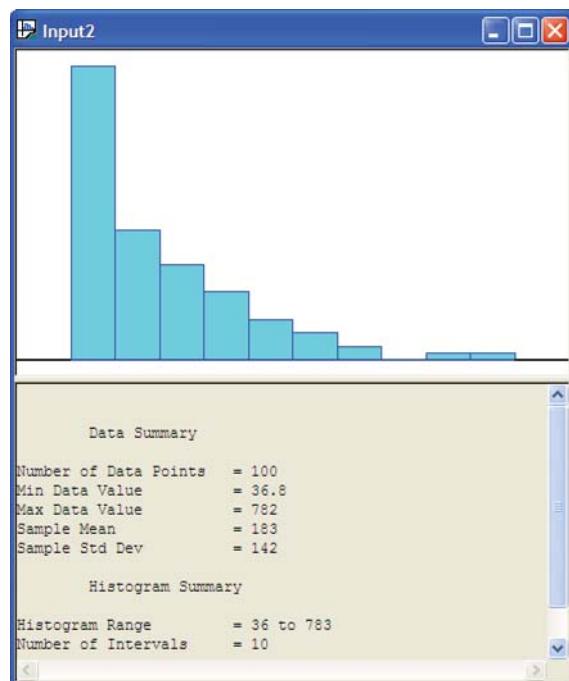
In this section, we use the input analyzer available within Arena<sup>TM</sup> to fit a distribution to service times collected for the pharmacy example. Let  $X_i$  be the service time of the  $i$ th customer, where the service time is defined as starting when the  $(i - 1)$ th customer begins to drive off and ending when the  $i$ th customer drives off after interacting with the pharmacist. In the case where there is no customer already in line when the  $i$ th customer arrives, the start of the service can be defined as the point where the customer's car arrives to the beginning of the space in front of the pharmacist's window. Notice that in this definition, the time that it takes the car to pull up to the pharmacy window is being included. An alternative definition of service time might simply be the time between when the pharmacist asks the customer what they need until the time in which the customer gets the receipt. Both of these definitions are reasonable interpretations of service times and it is up to you to decide what sort of definition fits best with the overall modeling objectives. As you can see, input modeling is as much an art as it is a science.

One hundred observations of the service time were collected using a portable digital assistant and are shown in Table 6.5 where the first observation is in row 1 column 1, the second observation is in row 2 column 1, the 21st observation is in row 1 column 2, and so forth. This data is available in the text file *PharmacyInputModelingExampleData.txt* that accompanies this chapter.

Prior to using the input analyzer, you should check the data if the observations are stationary (not dependent on time) and whether it is independent. We will leave that analysis as an exercise, since we have already illustrated the process in the previous section.

**TABLE 6.5 Pharmacy Service Times**

61	278.73	194.68	55.33	398.39
59.09	70.55	151.65	58.45	86.88
374.89	782.22	185.45	640.59	137.64
195.45	46.23	120.42	409.49	171.39
185.76	126.49	367.76	87.19	135.6
268.61	110.05	146.81	59	291.63
257.5	294.19	73.79	71.64	187.02
475.51	433.89	440.7	121.69	174.11
77.3	211.38	330.09	96.96	911.19
88.71	266.5	97.99	301.43	201.53
108.17	71.77	53.46	68.98	149.96
94.68	65.52	279.9	276.55	163.27
244.09	71.61	122.81	497.87	677.92
230.68	155.5	42.93	232.75	255.64
371.02	83.51	515.66	52.2	396.21
160.39	148.43	56.11	144.24	181.76
104.98	46.23	74.79	86.43	554.05
102.98	77.65	188.15	106.6	123.22
140.19	104.15	278.06	183.82	89.12
193.65	351.78	95.53	219.18	546.57

**Figure 6.21** Input analyzer after data import.

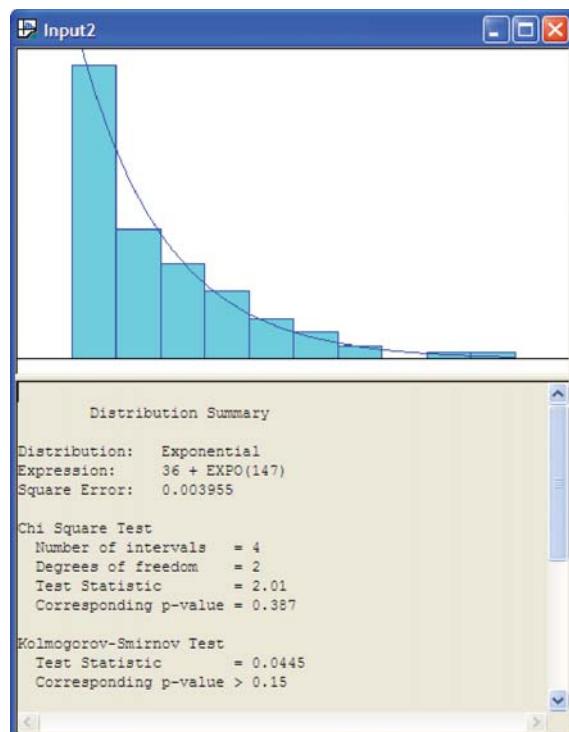
After opening the input analyzer, you should choose New from the File menu to start a new input analyzer data set. Then, using File > Data File > Use Existing, you can import the text file containing the data for analysis. The resulting import should leave the input analyzer looking similar to Figure 6.21.

You should save the session which will create a (.dft) file. Notice how the input analyzer automatically makes a histogram of the data and performs a basic statistical summary of the data. In looking at Figure 6.21, we might hypothesize a distribution that has long tail to the right, such as the exponential distribution.

The Input Analyzer will fit many of the common distributions that are available within Arena™: Beta, Erlang, Exponential, Gamma, Lognormal, Normal, Triangular, Uniform, Weibull, Empirical, Poisson. In addition, it will provide the Arena™ expression to be used within the Arena™ model. The fitting process within the Input Analyzer is highly dependent upon the intervals that are chosen for the histogram of the data. Thus, it is very important that you vary the number of intervals and check the sensitivity of the fitting process to the number of intervals in the histogram.

There are two basic methods by which you can perform the fitting process (i) individually for a specific distribution and (ii) by fitting all of the possible distributions. Given the interval specification, the Input Analyzer will compute a chi-squared goodness-of-fit statistic, K-S test, and squared error criteria, all of which will be discussed in what follows.

Let us try to fit an exponential distribution to the observations. With the formerly imported data imported into an input window within the Input Analyzer, go to the Fit menu and select the exponential distribution. The resulting analysis is shown in Listing 6.1 and



**Figure 6.22** Histogram for exponential fit to service times.

Figure 6.22. The Input Analyzer has made a fit to the data and has recommended the Arena expression ( $36 + \text{EXPO}(147)$ ). What is this value 36? The value 36 is called the offset or location parameter. Recall the discussion in Chapter 2 concerning shifted distributions. Any distribution can have this additional parameter, which further complicates parameter estimation procedures. The Input Analyzer has an algorithm that will attempt to estimate this parameter. Is the model reasonable for the service time data? From the histogram with the exponential distribution overlaid, it appears to be a reasonable fit.

To understand the results of the fit, you must understand how to interpret the results from the chi-square test and the K-S test. The null hypothesis is that the data come from the hypothesized distribution versus the alternative hypothesis is that the data do not come from the hypothesized distribution. The Input Analyzer shows the *p*-value of the tests.

Listing 6.1 indicates that the *p*-value for the chi-square test is 0.387. Thus, we would not reject the hypothesis that the service times come from the proposed exponential distribution. For the K-S test, the *p*-value is greater than 0.15 which also does not suggest a serious lack of fit for the exponential distribution.

Figure 6.23 shows the results of fitting a uniform distribution to the data. Listing 6.2 for the uniform distribution shows that the *p*-value for the K-S test is smaller than 0.01, which indicates that the uniform distribution is probably not a good model for the service times.

### **Listing 6.1 Exponential Summary from Input Analyzer**

```
Distribution Summary
Distribution: Exponential
Expression: 36 + EXPO(147)
Square Error: 0.003955

Chi Square Test
Number of intervals = 4
Degrees of freedom = 2
Test Statistic      = 2.01
Corresponding p-value = 0.387

Kolmogorov-Smirnov Test
Test Statistic      = 0.0445
Corresponding p-value > 0.15

Data Summary
Number of Data Points = 100
Min Data Value       = 36.8
Max Data Value       = 782
Sample Mean          = 183
Sample Std Dev        = 142

Histogram Summary
Histogram Range      = 36 to 783
Number of Intervals = 10
```

**Listing 6.2 Uniform Distribution Summary from Input Analyzer**

```

Distribution Summary
Distribution: Uniform
Expression: UNIF(36, 783)
Square Error: 0.156400

Chi Square Test
Number of intervals = 7
Degrees of freedom = 6
Test Statistic      = 164
Corresponding p-value < 0.005

Kolmogorov-Smirnov Test
Test Statistic      = 0.495
Corresponding p-value < 0.01

Data Summary
Number of Data Points = 100
Min Data Value       = 36.8
Max Data Value       = 782
Sample Mean          = 183
Sample Std Dev        = 142

Histogram Summary
Histogram Range      = 36 to 783
Number of Intervals = 10

```

In general, you should be cautious of goodness-of-fit tests because they are unlikely to reject any distribution when you have little data and they are likely to reject every distribution when you have lots of data. The point is, for whatever software that you use for your modeling fitting, you will need to correctly interpret the results of any statistical tests that are performed. Be sure to understand how these tests are computed and how sensitive the tests are to various assumptions within the model fitting process.

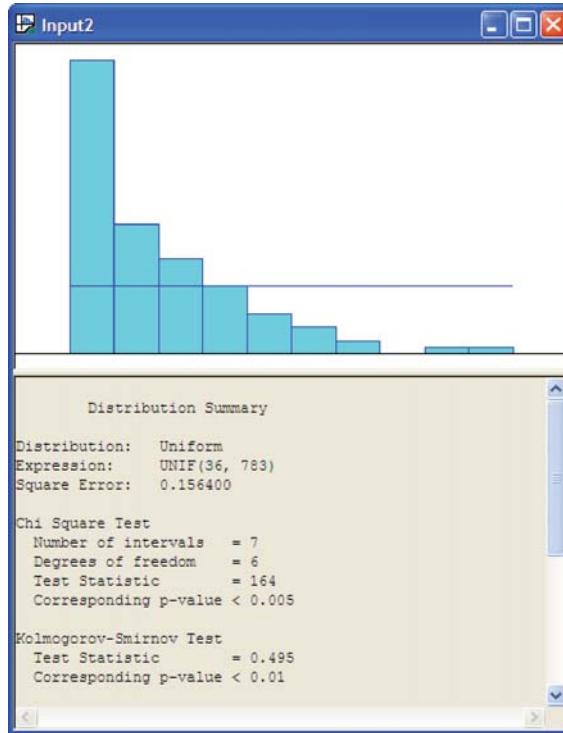
The final result of interest in the Input Analyzer's distribution summary output is the value labeled *Square Error*. This is the criteria that the Input Analyzer uses to recommend a particular distribution when fitting multiple distributions at one time to the data.

The squared error is defined as the sum over the intervals of the squared difference between the relative frequency and the probability associated with each interval:

$$\text{Square error} = \sum_{j=1}^k (h_j - \hat{p}_j)^2 \quad (6.32)$$

Table 6.6 shows the square error calculation for the fit of the exponential distribution to the service time data. The computed square error matches closely the value computed within the Input Analyzer, with the difference attributed to round off errors.

When you chose the Fit All option within the Input Analyzer, each of the possible distributions are fit in turn and the summary results computed. Then, the Input Analyzer ranks the distributions from smallest to largest according to the square error criteria. As you can



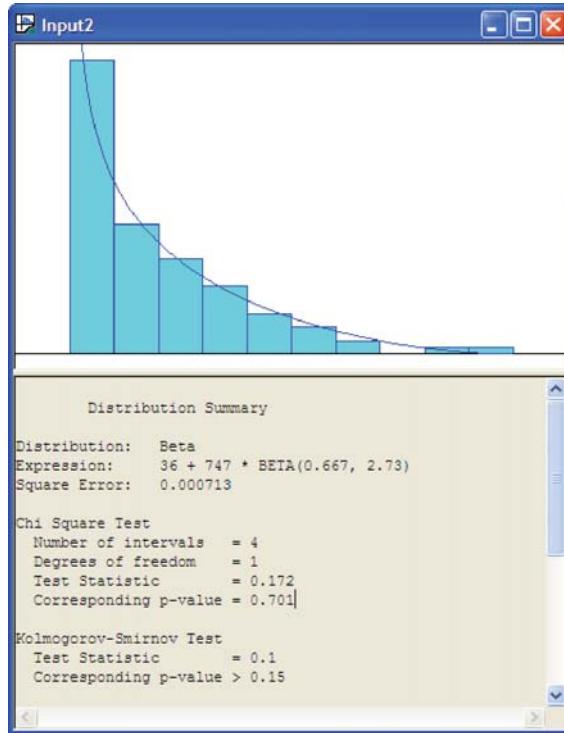
**Figure 6.23** Uniform distribution and histogram for service time data.

**TABLE 6.6 Square Error Calculation**

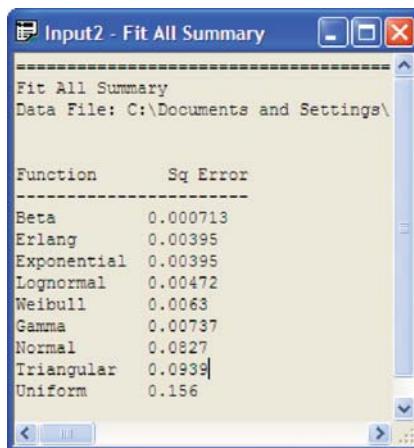
$j$	$c_j$	$b_j$	$h_j$	$\hat{p}_j$	$(h_j - \hat{p}_j)^2$
1	43	111	0.43	0.399	0.000961
2	19	185	0.19	0.24	0.0025
3	14	260	0.14	0.144	1.6E-05
4	10	335	0.1	0.0866	0.00018
5	6	410	0.06	0.0521	6.24E-05
6	4	484	0.04	0.0313	7.57E-05
7	2	559	0.02	0.0188	1.44E-06
8	0	634	0	0.0113	0.000128
9	1	708	0.01	0.0068	1.02E-05
10	1	783	0.01	0.00409	3.49E-05
Square error					0.003969

see from the definition of the square error criteria, the metric is dependent upon the defining intervals. Thus, it is highly recommended that you test the sensitivity of the results to different values for the number of intervals.

Using the Fit All function results in the Input Analyzer suggesting that  $36 + 747 * \text{BETA}(0.667, 2.73)$  expression is a good fit of the model (Figure 6.24). The Window > Fit All Summary menu option will show the squared error criteria for all the distributions that were fit. The summary indicates that the exponential distribution is second in the fitting



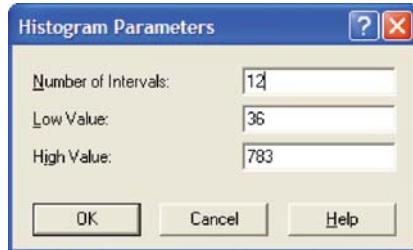
**Figure 6.24** Fit all recommendation.



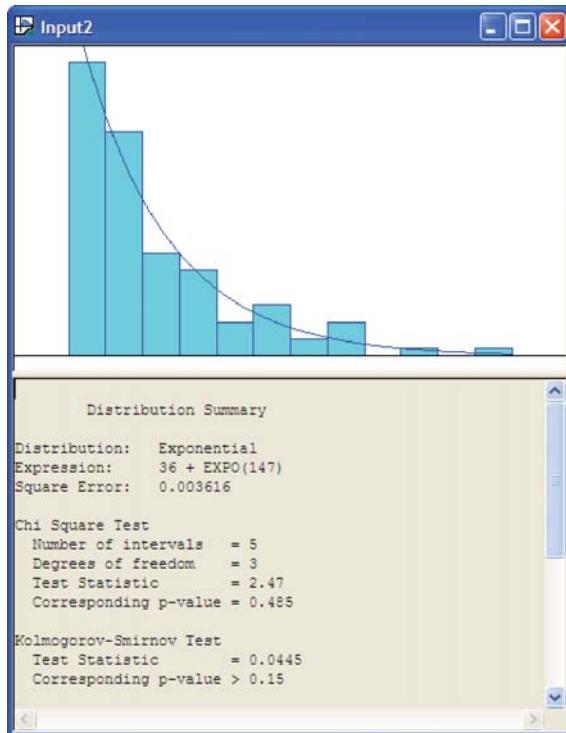
**Figure 6.25** Summary of squared error.

process according to the squared error criteria (Figure 6.25). By using Options > Parameters > Histogram, the Histogram Parameters dialog can be used to change the parameters associated with the histogram as shown in Figure 6.26.

Changing the number of intervals to 12 results in the output provided in Figure 6.27, which indicates that the exponential distribution is a reasonable model based on the



**Figure 6.26** Changing the histogram parameters.

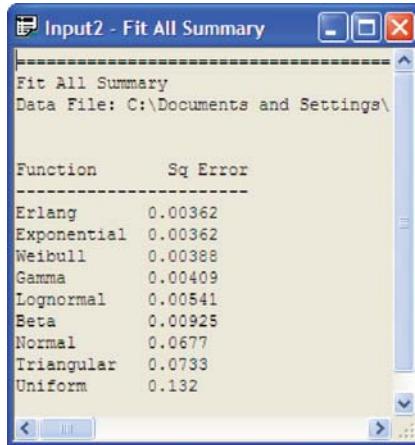


**Figure 6.27** Fit All with 12 intervals.

chi-square test, the K-S test, and the squared error criteria. You are encouraged to check other fits with differing number of intervals. In most of the fits, the exponential distribution will be recommended. It is beginning to look like the exponential distribution is a reasonable model for the service time data.

The Input Analyzer is convenient because it has the fit all summary and will recommend a distribution (Figure 6.28). However, it does not provide P-P plots and Q-Q plots. To do this, we can use the *fitdistrplus* package within R. Before proceeding with this analysis, there is a technical issue that must be addressed.

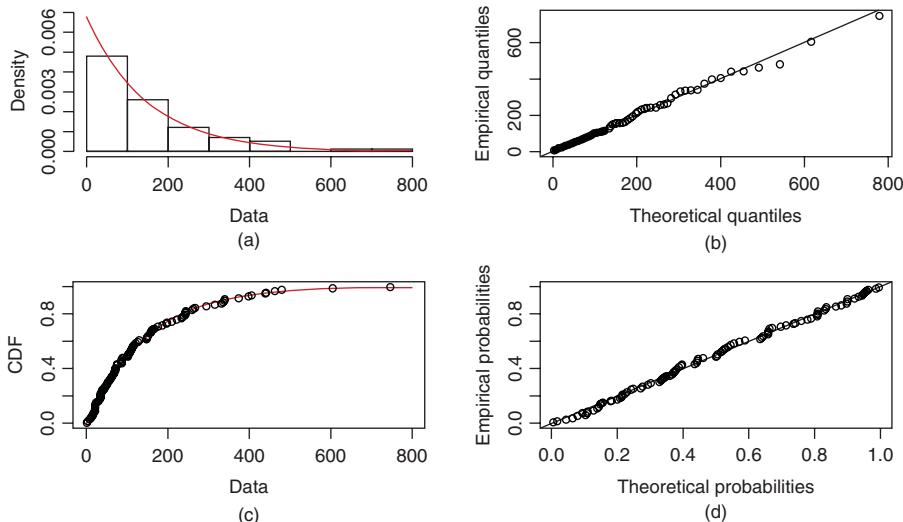
The proposed model from the Input Analyzer is:  $36 + \text{EXPO}(147)$ . That is, if  $X$  is a random variable that represents the service time, then  $X \sim 36 + \text{EXPO}(147)$ , where 147 is



**Figure 6.28** Fit All summary with 12 intervals.

the mean of the exponential distribution, so that  $\lambda = 1/147$ . Since 36 is a constant in this expression, this implies that the random variable  $W = X - 36$  has  $W \sim \text{EXPO}(147)$ . Thus, the model checking versus the exponential distribution can be done on the random variable  $W$ . That is, take the original data and subtract 36.

The following listing illustrates the R commands to make the fit, assuming that the data is in a file called *ServiceTimes.txt* within the R working directory. Figure 6.29 shows that the exponential distribution is a good fit for the service times based on the empirical distribution, P–P plot, and the Q–Q plot.



**Figure 6.29** Fit distribution plot from *fitdistrplus* for exponential distribution fit. (a) Empirical and theoretical density, (b) Q–Q plot, (c) empirical and theoretical CDFs, and (d) P–P plot.

```

> x = scan(file="ServiceTimes.txt") #read in the file
Read 100 items
> w=x-36
> library(fitdistrplus)
Loading required package: survival
Loading required package: splines
> fw = fitdist(w, "exp")
> fw
Fitting of the distribution ' exp ' by maximum likelihood
Parameters:
      estimate   Std. Error
rate 0.006813019 0.0006662372
> 1/fw$estimate
      rate
146.7778
> plot(fw)

```

## 6.8 ADDITIONAL INPUT MODELING CONCEPTS

This section wraps up the discussion of input modeling by covering some additional topics that often come up during the modeling process.

Throughout the service time example, a continuous random variable was being modeled. But what do you do if you are modeling a discrete random variable? The basic complicating factor is that the only discrete distribution available within the Input Analyzer is the Poisson distribution and this option will only become active if the data input file only has integer values. The steps in the modeling process are essentially the same except that you cannot rely on the Input Analyzer. Commercial software will have options for fitting some of the common discrete distributions. The fitting process for a discrete distribution is simplified in one way because the bins for the frequency diagram are naturally determined by the range of the random variable. For example, if you are fitting a geometric distribution, you need only to tabulate the frequency of occurrence for each of the possible values of the random variable 1, 2, 3, 4, etc.. Occasionally, you may have to group bins together to get an appropriate number of observations per bin. The fitting process for the discrete case primarily centers on a straightforward application of the chi-squared goodness-of-fit test, which was outlined in this chapter, and is also covered in many introductory probability and statistics textbooks.

If you consider the data set as a finite set of values, then why cannot you just reuse the data? In other words, why should you go through all the trouble of fitting a theoretical distribution when you can simply reuse the observed data, say, for example, by reading in the data from a file. There are a number of problems with this approach. The first difficulty is that the observations in the data set are a *sample*. That is, they do not necessarily cover all the possible values associated with the random variable. For example, suppose that only a sample of 10 observations was available for the service time problem. Furthermore, assume that the sample was as follows:

1	2	3	4	5	6	7	8	9	10
36.84	38.5	46.23	46.23	46.23	48.3	52.2	53.46	53.46	56.11

Clearly, this sample does not contain the high service times that were in the 100 sample case. The point is, if you resample from only these values, you will never get any values less than 36.84 or bigger than 56.11. The second difficulty is that it will be more difficult to experimentally vary this distribution within the simulation model. If you fit a theoretical distribution to the data, you can vary the parameters of the theoretical distribution with relative ease in any experiments that you perform. Thus, it is worthwhile to attempt to fit and use a reasonable input distribution.

But what if you cannot find a reasonable input model either because you have very limited data or because no model fits the data very well? In this situation, it is useful to try to use the data in the form of the empirical distribution. Essentially, you treat each observation in the sample as equally likely and randomly draw observations from the sample. In many situations, there are repeated observations within the sample (as above) and you can form a discrete empirical distribution over the values. If this is done for the sample of 10 data points, a discrete empirical distribution can be formed as shown in Table 6.7. Within Arena<sup>TM</sup>, this distribution can be modeled using the *DISC()* function. Again, this limits us to only the values observed in the sample.

One can also use the continuous empirical distribution, which interpolates between the distribution values. In Arena<sup>TM</sup>, this is represented with the function *CONT()*. The Input Analyzer's empirical function will fit either a discrete empirical or a continuous empirical to the data set.

What do you do if the analysis indicates that the data is dependent or that the data is non-stationary? Either of these situations can invalidate the basic assumptions behind the standard distribution fitting process. First, suppose that the data shows some correlation. The first thing that you should do is to verify that the data was correctly collected. The sampling plan for the data collection effort should attempt to ensure the collection of a random sample. If the data was from an automatic collection procedure, then it is quite likely that there may be correlation in the observations. This is one of the hazards of using automatic data collection mechanisms. You then need to decide whether modeling the correlation is important or not to the study at hand. Thus, one alternative is to simply ignore the correlation and to continue with the model fitting process. This can be problematic for two reasons. First, the statistical tests within the model fitting process will be suspect, and second, the correlation may be an important part of the input modeling. For example, it has been shown that correlated arrivals and correlated service times in a simple queuing model can have significant effects on the values of the queue's performance measures. If you have a large enough data set, a basic approach is to form a random sample from the

**TABLE 6.7 Simple Empirical Distribution**

X	PMF	CDF
36.84	0.1	0.1
38.5	0.1	0.2
46.23	0.3	0.5
48.3	0.1	0.6
53.46	0.2	0.8
55.33	0.1	0.9
56.11	0.1	1

**TABLE 6.8 Breakpoint-Based Empirical Distribution**

X	PMF	CDF
(36, 100]	0.1	0.1
(100, 200]	0.1	0.2
(200, 400]	0.3	0.6
(400, 600]	0.2	0.8
(600, $\infty$ )	0.2	1.0

data set itself in order to break up the correlation. Then, you can proceed with fitting a distribution to the random sample; however, you should still model the dependence by trying to incorporate it into the random generation process. There are some techniques for incorporating correlation into the random variable generation process which are discussed in Chapter 10.

If the data show non-stationary behavior, then you can attempt to model the dependence on time using time series models or other non-stationary models. Suffice to say that these advanced techniques are beyond the scope of this text; however, the next section will discuss the modeling of a special non-stationary model, the nonhomogeneous Poisson process, which is very useful for modeling time dependent arrival processes. For additional information on these methods, the interested reader is referred to Law [2007] and Leemis and Park [2006] or the references therein.

Finally, all of the above assumes that you have data from which you can perform an analysis. In many situations, you might have no data whatsoever either because it is too costly to collect or because the system that you are modeling does not exist. In the latter case, you can look at similar systems and see how their inputs were modeled, perhaps adopting some of those input models for the current situation. In either case, you might also rely on expert opinion. In this situation, you can ask an expert in the process to describe the characteristics of a probability distribution that might model the situation. This is where the uniform and the triangular distributions can be very useful, since it is relatively easy to get an expert to indicate a minimum possible value, a maximum possible value, and even a most likely value.

Alternatively, you can ask the expert to assist in making an empirical distribution based on providing the chance that the random variable falls within various intervals. The breakpoints near the extremes are especially important to get. Table 6.8 presents a distribution for the service times based on this method.

Whether you have lots of data, little data, or no data, the key final step in the input modeling process is *sensitivity analysis*. Your ultimate goal is to use the input models to drive your larger simulation model of the system under study. You can spend significant time and energy collecting and analyzing data for an input model that has no significant effect on the output measures of interest to your study. You should start out with simple input models and incorporate them into your simulation model. Then, you can vary the parameters and characteristics of those models in an experimental design to assess how sensitive your output is to the changes in the inputs. If you find that the output is very sensitive to particular input models, then you can plan, collect, and develop better models for those situations. The amount of sensitivity is entirely modeler dependent. Remember that in this whole process, you are in charge, not the software. The software is only there to support your decision-making process. Use the software to justify your art.

## 6.9 MODELING RANDOMNESS IN ARENA

In this section, we get back to Arena<sup>TM</sup> and illustrate the use of input distributions within a larger model.

The previous sections have discussed a number of methods and functions for generating random variables. This section presents an example to illustrate the usage of input distributions in Arena<sup>TM</sup>. The example is fictitious but has a number of interesting features. A variety of probability distributions are involved to illustrate their use.

### EXAMPLE 6.4 LOTR Makers, Inc.

Every morning the sales force at LOTR Makers, Inc. makes a number of confirmation calls to customers who have previously been visited by the sales force. They have tracked the success rate of their confirmation calls over time and have determined that the chance of success varies from day to day. They have modeled the probability of success for a given day as a beta random variable with parameters  $\alpha_1 = 5$  and  $\alpha_2 = 1.5$  so that the mean success rate is about 77%. They always make 100 calls every morning. Each sales call will or will not result in an order for a pair of magical rings for that day. Thus, the number of pairs of rings to produce every day is a binomial random variable, with  $p$  determined by the success rate for the day and  $n = 100$  representing the total number of calls made. Note that  $p$  is random in this modeling.

The sale force is large enough and the time to make the confirmation calls small enough so as to be able to complete all the calls before releasing a production run for the day. In essence, ring production does not start until all the orders have been confirmed, but the actual number of ring pairs produced every day is unknown until the sales call confirmation process is completed. The time to make the calls is negligible when compared to the overall production time.

Besides being magical, one ring is smaller than the other ring so that the smaller ring must fit snuggly inside the larger ring. The pair of rings is produced by a master ring maker and takes uniformly between 5 and 15 minutes. The rings are then scrutinized by an inspector with the time (in minutes) being distributed according to a triangular distribution with parameters (2, 4, 7) for the minimum, the mode, and the maximum. The inspection determines whether the smaller ring is too big or too small when fit inside the bigger outer ring. The inside diameter of the bigger ring,  $D_b$ , is normally distributed with a mean of 1.5 centimeter and a standard deviation of 0.002. The outside diameter of the smaller ring,  $D_s$ , is normally distributed with a mean of 1.49 centimeter and a standard deviation of 0.005. If  $D_s > D_b$ , then the smaller ring will not fit in the bigger ring; however, if  $D_b - D_s > tol$ , then the rings are considered too loose. The tolerance is currently set at 0.02 centimeter.

If there are no problems with the rings, the rings are sent to a packer for custom packaging for shipment. A time study of the packaging time indicates that it is distributed according to a lognormal distribution with a mean of 7 minutes and a standard deviation of 1 minute. If the inspection shows that there is a problem with the pair of rings, they are sent to a rework craftsman. The minimum time that it takes to rework the pair of rings has been determined to be 5 minutes plus some random time that is distributed according to a Weibull distribution with a scale parameter of 15 and a shape parameter of 5. After the rework is completed, the pair of rings is sent to packaging.

LOTR Makers, Inc. is interested in estimating the daily production time. In particular, management is interested in estimating the probability of overtime. Currently, the company runs two shifts of 480 minutes each. Time after the end of the second shift is considered overtime. Use 30 simulated days to investigate the situation.

### 6.9.1 Conceptualizing the Model

Now let us proceed with the modeling of Example 6.4. We start with answering the basic model building questions.

- *What is the system? What information is known by the system?*

The system is the LOTR Makers, Inc. sales calls and ring production processes. The system starts every day with the initiation of sales calls and ends when the last pair of rings produced for the day is shipped. The system knows the following:

- Sales call success probability distribution:  $p \sim \text{BETA}(\alpha_1 = 5, \alpha_2 = 1.5)$
- Number of calls to be made every morning:  $n = 100$
- Distribution of time to make the pair of rings:  $\text{UNIF}(5,15)$
- Distributions associated with the big and small ring diameters:  $\text{NORM}(1.5, 0.002)$  and  $\text{NORM}(1.49, 0.005)$ , respectively
- Distribution of ring-inspection time:  $\text{TRIA}(2,4,7)$
- Distribution of packaging time:  $\text{LOGN}(7,1)$
- Distribution of rework time,  $5 + \text{WEIB}(15, 3)$
- Length of a shift: 480 minutes

- *What are the entities? What information must be recorded for each entity?*

Possible entities are the sales calls and the production job (pair of rings) for every successful sales call. Each sales call knows whether it is successful. For every pair of rings, the diameters must be known.

- *What are the resources that are used by the entities?*

The sales calls do not use any resources. The production job uses a master craftsman, an inspector, and a packager. It might also use a rework craftsman.

- *What are the process flows? Write out or draw sketches of the process.*

There are two processes: sales order and production. An outline of the sales order process should look like the following:

1. Start the day.
2. Determine the likelihood of calls being successful.
3. Make the calls.

4. Determine the total number of successful calls.
5. Start the production jobs.

An outline of the production process should look like the following:

1. Make the rings (determine sizes).
2. Inspect the rings.
3. If rings do not pass inspection, perform rework.
4. Package rings and ship.

The next step in the modeling process is to develop pseudo-code for the situation.

Each of the above-described processes needs to be modeled. Note that in the problem statement, there is no mention that the sales calls take any significant time. In addition, the sales order process does not use any resources. The calls take place before the production shifts. The major purpose of the sales order process is to determine the number of rings to produce for the daily production run. This type of situation is best modeled using a logical entity. In essence, the logical entity represents the entire sales order process and must determine the number of rings to produce. From the problem statement, this is a binomial random variable with  $n = 100$  and  $p \sim \text{BETA}(\alpha_1 = 5, \alpha_2 = 1.5)$ . Arena™ does not have a binomial distribution. The easiest way to accomplish this is to use the convolution property of Bernoulli random variables to generate the binomial random variable.

In Exhibit 6.1, pseudo-code for the sales order process, the logical entity is first created and then assigned the values of  $p$ . Then, 100 Bernoulli random variables are generated to represent the success (or not) of a sales call. The successes are summed so that the appropriate number of production jobs can be determined. In the pseudo-code, the method of summing up the number of successes is done with a looping flow of control construct.

---

#### **Exhibit 6.1 Sales Order Process**

---

```

CREATE 1 order process logical entity
ASSIGN
    vSalesProb = BETA (5, 1.5)
    myCallNum = 1
END ASSIGN
WHILE myCallNum ≤ 100
    myCallNum = myCallNum + 1
    mySale = DISC(vSalesProb,1,1.0,0)
    myNumSales = myNumSales + mySale
END WHILE
SEPARATE
    create myNumSales duplicates, send to production
    dispose of original order logical entity
END SEPARATE

```

---

The production process pseudo-code describes what happens to a pair of rings as it moves through production (Exhibit 6.2).

---

**Exhibit 6.2 Production Process**

---

```

PROCESS
    SEIZE master ring maker
    DELAY for ring making
    RELEASE master ring maker
END PROCESS
ASSIGN
    myIDBigRing = NORM(1.49, 0.005)
    myODSmallRing = NORM(1.5, 0.002)
    myGap = myIDBigRing - myODSmallRing
END ASSIGN
PROCESS
    SEIZE inspector
    DELAY for inspection
    RELEASE inspector
END PROCESS
DECIDE
    IF myODSmallRing > myIDBigRing THEN
        RECORD too big
        Send to Rework
    ELSE
        RECORD not too big
        IF myGap > 0.02 THEN
            RECORD too small
            Send to Rework
        ELSE
            RECORD not too small
        END IF
    END IF
END DECIDE
PROCESS Packaging
    SEIZE packager
    DELAY for packaging
    RELEASE packager
END PROCESS
DISPOSE
A: REWORK
PROCESS
    SEIZE rework craftsman
    DELAY for rework
    RELEASE rework craftsman
END PROCESS
Send to Packaging

```

---

### 6.9.2 Implementing the Model

Thus far, the system has been conceptualized in terms of entities, resources, and processes. This provides a good understanding of the information that must be represented in the simulation model. In addition, the logical flow of the model has been represented in pseudo-code. Now it is time to implement these ideas in Arena™. The following describes an implementation for these processes. Before proceeding, you might want to try to implement the logic yourself so that you can check how you are doing against what is presented here. If not, you should try to implement the logic as you proceed through the example.

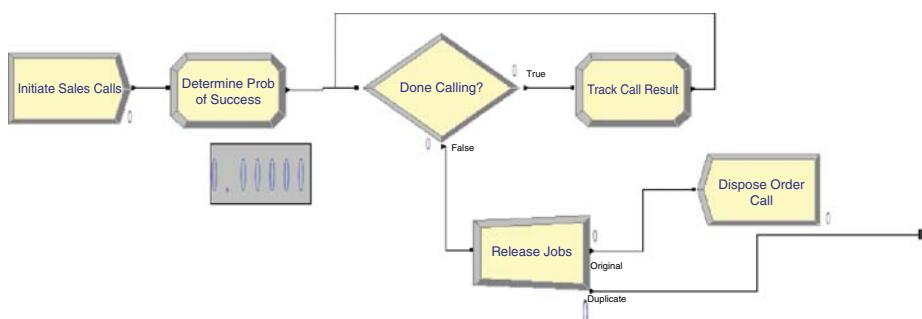
To implement the sales order process, you first need to decide how to represent the required data. In the pseudo-code, there is only one entity and no passage of time. In this situation, only one entity is needed to execute the associated modules. Thus, either variables or attributes can be used to implement the logic. A variable can be used because there is no danger of another entity changing the global value of the represented variables. In this model, the looping has been implemented using a looping DECIDE construct as discussed in Chapter 5. An overview of the sales order portion of the model is given in Figure 6.30. If you are building the model as you follow along, you should take the time to lay down the modules shown in the figure. Each of the modules will be illustrated in what follows.

The CREATE module is very straightforward. Because the Max Arrivals field is 1 and the First Creation field is 0.0, only one entity will be created at time 0.0. The fields associated with the label Time between Arrivals are irrelevant in this situation since there will not be any additional arrivals generated from this CREATE module. Fill out your CREATE module as shown in Figure 6.31.

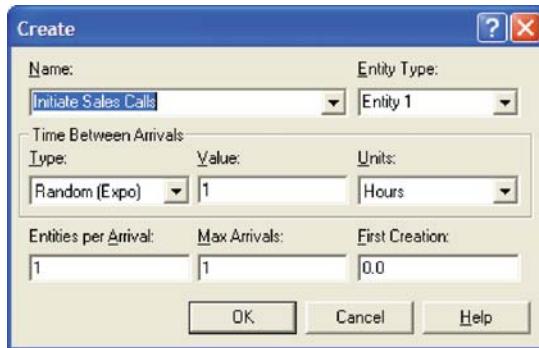
In the first ASSIGN module, the probability of success for the day is determined. The variable (*vSalesProb*) has been used to represent the probability as drawn from the BETA() distribution function (Figure 6.32). The attribute *myCallNum* is initialized to 1, prior to starting the looping logic. This attribute is going to count the number of calls made.

The DECIDE module uses the attribute *myCallNum* to check the variable *vNumCalls* (Figure 6.33). The variable *vNumCalls* is defined in the VARIABLE module (not shown here) and has been initialized to 100. Because the total number of calls has been represented as a variable, the desired number of calls can be easily changed in the VARIABLE module, without editing the DECIDE module.

Figure 6.34 presents the ASSIGN module in the looping logic. In this ASSIGN module, the attribute *myCallNum* is incremented. Then, according to the pseudo-code, you must implement a Bernoulli trial. This can easily be accomplished by using the DISC() distribution function with only two outcomes, 1 and 0, to assign to the attribute *mySale*. You can set the probability of success for the value 1 to *vSalesProb*, which was determined via the call to the BETA function. Note that the function DISC(BETA(5, 1.5), 1, 1.0, 0) would not work in this situation. DISC(BETA(5, 1.5)) will draw a new probability from the BETA() function every time DISC() is called. Thus, each trial will have a different probability of success. This is not what is required for the problem and this is why the variable *vSaleProb* was determined outside the looping logic. The attribute *myNumSales* is incremented with the value from *mySale*, which essentially counts up all the successful sales calls (marked with 1).



**Figure 6.30** Sales confirmation process.



**Figure 6.31** Initiating the sales call confirmation process.

Assignments				
	Type	Variable Name	Attribute Name	
1	Variable	vSalesProb	myOrderProb	BETA(5,1.5)
2	Attribute	Variable 2	myCallNum	1

**Figure 6.32** Determine the probability of success.

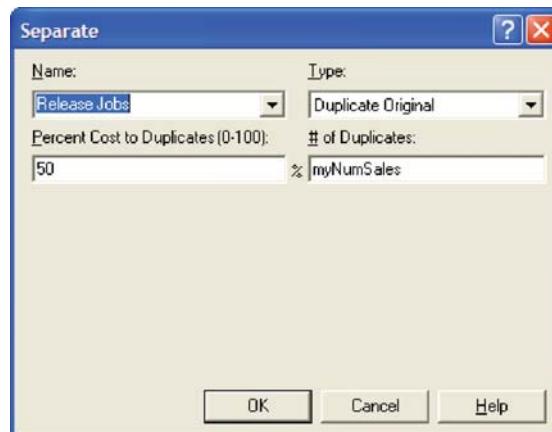
**Figure 6.33** Checking that all calls have been made.

	Type	Attribute Name	New Value
1	Attribute	myCallNum	myCallNum + 1
2	Attribute	mySale	DISC(vSalesProb,1,1,0,0)
3	Attribute	myNumSales	myNumSales + mySale

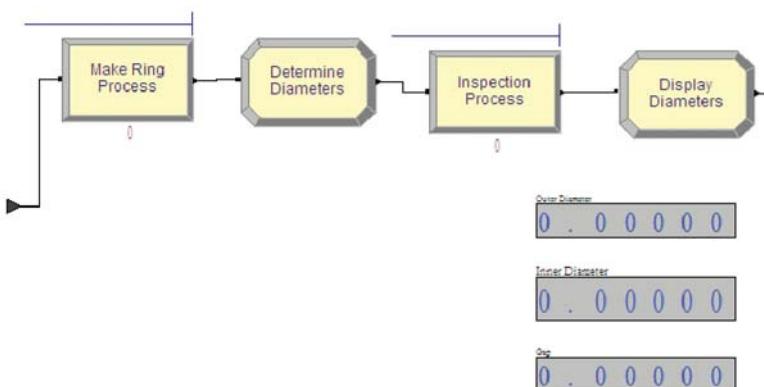
**Figure 6.34** Tracking the call's result.

When the DECIDE module evaluates to false, all the sales calls have been made, and the attribute *myNumSales* has recorded how many successful calls there have been out of the 100 calls made. The order process entity is directed to the SEPARATE module where it creates (*myNumSales*) duplicates and sends them to the production process (Figure 6.35). The original order process logic entity is sent to a DISPOSE module, which has its collect entity statistics box unchecked (not shown). By not checking this box, no statistics will be collected for entities that exit through this DISPOSE module. Since the problem does not require anything about the statistics for the order process logic entity, this is advisable.

The production process is too long for one screen shot, so this discussion is divided into three parts: the making and inspection processes, the decision concerning too big/too small, and the repair and packaging operations. Figure 6.36 presents the making and inspection processes which are implemented with PROCESS modules using the SEIZE, DELAY, RELEASE option. Figures 6.37 and 6.38 show the PROCESS modules and the appropriate delay distributions using the uniform and triangular distributions.



**Figure 6.35** Creating the jobs for production.



**Figure 6.36** Making and inspecting the rings.

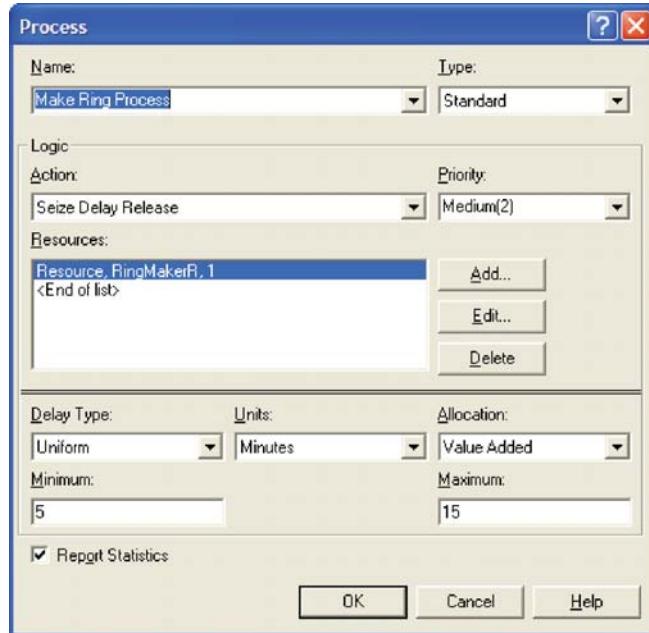


Figure 6.37 PROCESS module for making the rings.

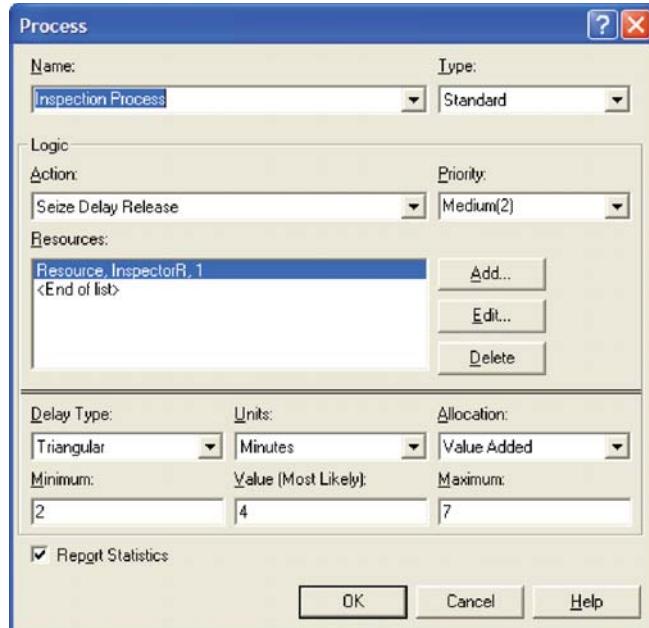


Figure 6.38 PROCESS module for inspecting the rings.

The ASSIGN module between the two processes is used to determine the diameters of the rings. Figure 6.39 shows that both *myIDBigRing* and *myODSmallRing* attributes are set using the normal distribution. The parameters of the NORM() functions are defined by variables in the VARIABLE module (not shown here).

After inspection, the rings must be checked to see whether rework is necessary. An overview of this checking is given in Figure 6.40. The RECORD modules are used to collect statistics on the probability of the smaller ring being too big or the smaller ring being too small.

Figure 6.41 shows the DECIDE module for checking if the ring is too big. The 2-way by condition option was used to check if the attribute (*myODSmallRing*) is larger than the attribute (*myIDBigRing*). If the smaller ring's outer diameter is larger than the bigger ring's inner diameter, then the smaller ring will not fit inside the bigger ring and the rings will require rework.

Figure 6.42 shows the DECIDE module for checking if the smaller ring is too loose. In this DECIDE module, the 2-way by condition option is used to check the expression (*myIDBigRing* - *myODSmallRing*) > *vTol*. If the difference between the diameters of the rings is too large (larger than the tolerance), then the rings are too loose and need rework. If the rings fit properly, then they go directly to packaging.

Figure 6.43 shows the rework and packaging processes. Again, the PROCESS module is used to represent these processes. Figures 6.44 and 6.45 show the rework and packaging PROCESS modules. Note that the delay type has been changed to expression so that 5 + WEIB(15, 3) and LOGN(7, 1) can be specified for each of the respective processing times.

Assignments				
	Type	Attribute Name	New Value	
1	Attribute	myIDBigRing	NORM(vIDM, vIDS)	
2	Attribute	myODSmallRing	NORM(vODM, vODS)	
3	Attribute	myGap	myIDBigRing - myODSmallRing	

Double-click here to add a new row.

Figure 6.39 Determining the ring diameters.

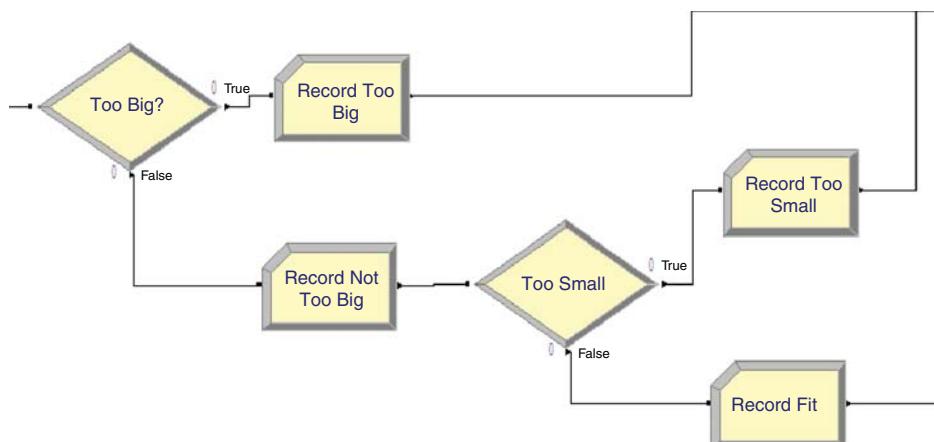
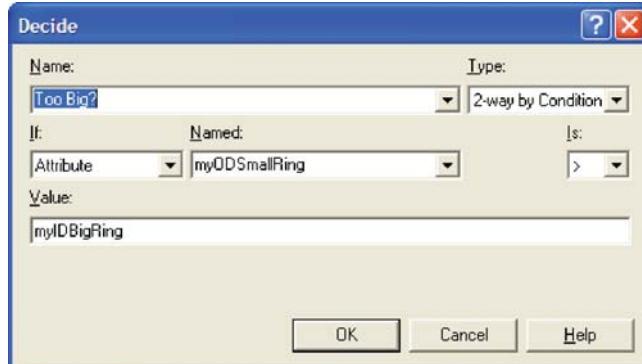
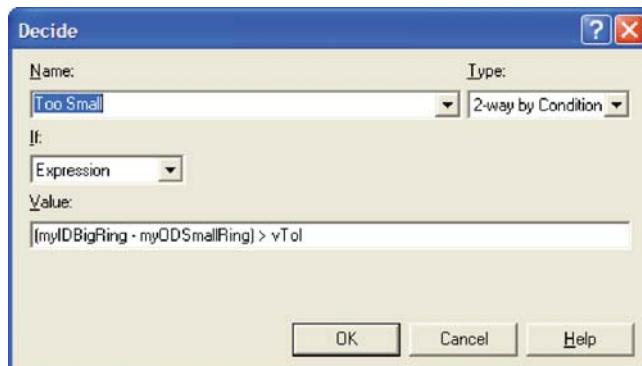


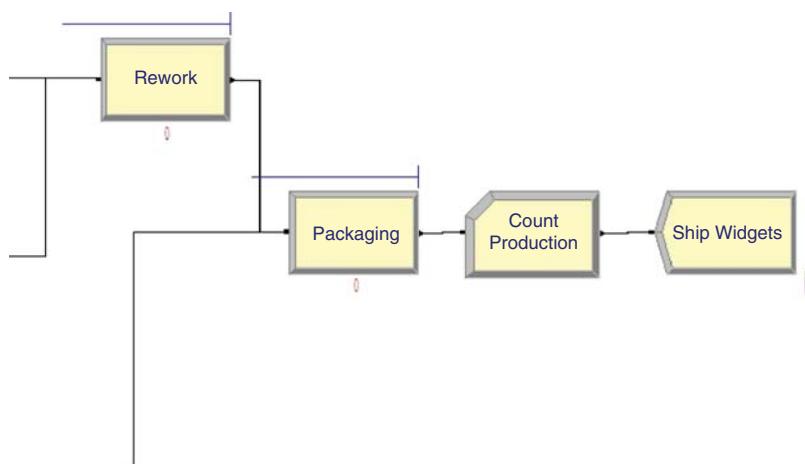
Figure 6.40 Checking ring diameters.



**Figure 6.41** Checking to determine whether small ring is too big.



**Figure 6.42** Checking to determine whether smaller ring is too small.



**Figure 6.43** Rework and packaging processes.

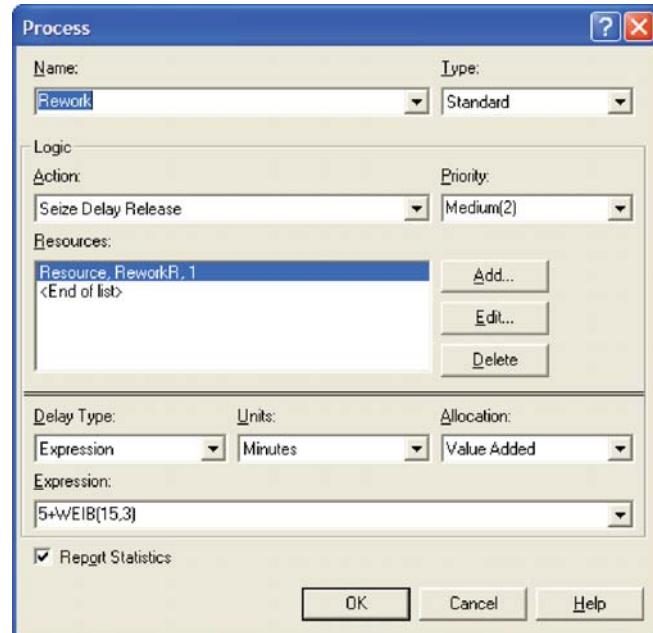


Figure 6.44 Rework PROCESS module.

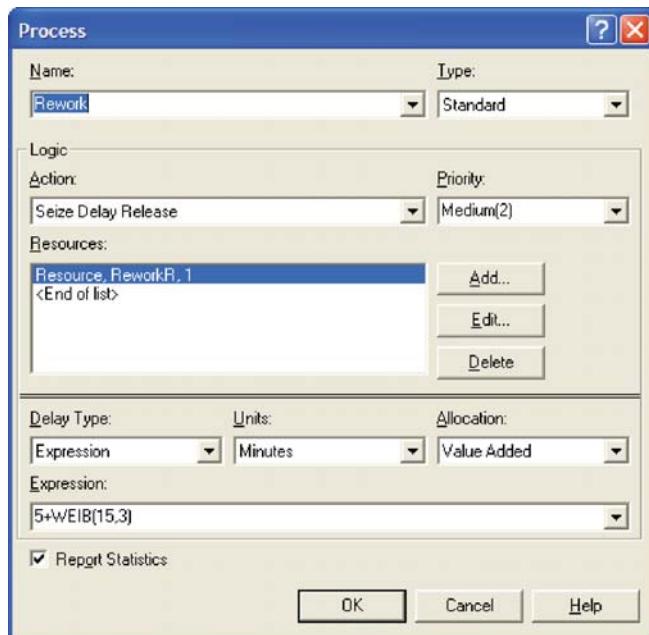


Figure 6.45 Packaging PROCESS module.

In Arena™, a simulation can end based on three situations:

1. A scheduled run length
2. A terminating condition is met
3. No more events (entities) to process.

The problem statement requests the estimation of the probability of overtime work. The sales order process determines the number of rings to produce. The production process continues until there are no more rings to produce for that day. The number of rings to produce is a binomial random variable as determined by the sales order confirmation process. Thus, there is no clear run length for this simulation.

Because the production for the day stops when all the rings are produced, the third situation applies for this model. The simulation will end automatically after all the rings are produced. In essence, a day's worth of production is simulated. Because the number of rings to produce is random and it takes a random amount of time to make, inspect, rework, and pack the rings, the time that the simulation will end is a random variable. If this time is less than 960 minutes (the time of two shifts), there will not be any overtime. If this time is greater than 960 minutes, production will have lasted past two shifts and thus overtime will be necessary. To assess the chance that there is overtime, you need to record statistics on how often the end of the simulation is past 960 minutes.

The special variable *TNOW* provides the current time of the simulation. Thus, when the simulation ends, *TNOW* will be the time that the simulation ended. In this case, *TNOW* will represent the time that the last ring completed processing. To estimate the chance that there is overtime, the Advanced Process panel's statistic module can be used. In particular, you need to define what is called an OUTPUT statistic.

An OUTPUT statistic records the final value of some system or statistical value at the end of a replication. An OUTPUT statistic can be defined by any valid expression involving system variables, variables, statistical functions, and so on. In Figure 6.46, an OUTPUT statistic called *TimeToMakeOrderStat* has been defined, which records the value of *TNOW*. This will cause the collection of statistics (across replication) for the ending value of *TNOW*. The OUTPUT statistic will record the average, minimum, maximum, and half-width across the replications for the defined expression. You can also use the Output File field to write out the observed values to a file if necessary.

The second OUTPUT statistic in Figure 6.46 defines an OUTPUT statistic called *ProbOfOT* to represent the chance that the production lasts longer than 960 minutes. In this case, the expression is the Boolean value of *TNOW > 960*. The expression *TNOW > 960* will be

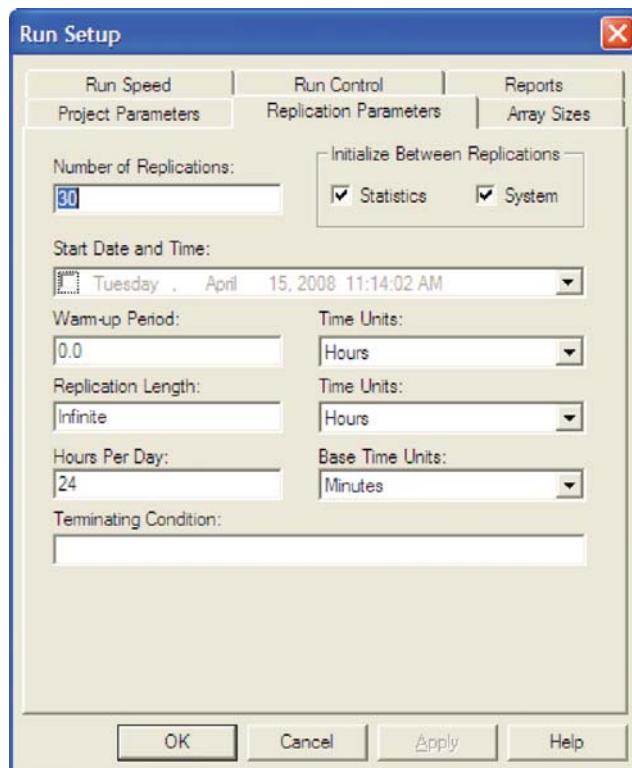
Statistic - Advanced Process					
	Name	Type	Expression	Report Label	Output File
1	TimeToMakeOrderStat	Output	TNOW	TimeToMakeOrderStat	[...]
2	ProbOfOT	Output	TNOW > 960	ProbOfOT	
Double-click here to add a new row.					

**Figure 6.46** Defining OUTPUT statistics for overtime.

evaluated. If it is true, it will evaluate to 1.0; otherwise, it will evaluate to a 0.0. This is just like an indicator variable on the desired condition. The OUTPUT statistic will compute the average of the 1's and 0's, which is an estimate of the probability of the condition. Thus, you will get an estimate of the likelihood of overtime.

Figure 6.47 shows how to set up the run parameters of the simulation. You should run the simulation with a base time unit of minutes. This is important to set up here so that  $TNOW$  can now be interpreted in minutes. This ensures that the expression  $TNOW > 960$  makes sense in terms of the desired units. There is no replication length specified or there is no terminating condition specified. As previously mentioned, the simulation ends when there are no more events (entities) to process. If the model is not set up correctly (i.e., an infinite number of entities are processed), then the simulation will never terminate. This is a logical situation that the modeler is responsible for preventing. The number of replications has been specified to 30 to represent the 30 days of production.

Running the model results in the user-defined statistics for the probability of overtime and the average time to produce the orders as shown in Figure 6.48. The probability of overtime appears to be about 6%, but the 95% half-width is wide for these 30 replications. The average time to produce an order is about 780.68 minutes. While the average is under 960, there still appears to be a reasonable chance of overtime occurring. In the exercises,



**Figure 6.47** Specifying the number of replications in Run Setup.

Output	Average	Half Width	Minimum Average	Maximum Average
ProbOfOT	0.06666667	0.09	0.00	1.0000
TimeToMakeOrderStat	780.68	49.12	462.57	968.44

**Figure 6.48** Output statistics across 30 days.

you are asked to explore the reasons behind the overtime and to recommend an alternative to reduce the likelihood of overtime.

In this example, you have learned how to model a small system involving random components and to translate the conceptual model into an Arena™ simulation. Using the simulation, you can then observe the statistical responses associated with the model. Chapter 7 will concentrate on how to properly collect and analyze the statistical responses from such simulations.

In addition, you have learned how to use many of the probability distributions available in Arena™ to generate random variables for the LOTR Makers, Inc. situation. There are only a couple of distributions that have not been covered through the previous examples (e.g., empirical continuous CONT(), Johnson JOHN(), Poisson POIS(), and nonstationary exponential NSExpo()). Chapter 10 covers the NSExpo() function, and an exercise will allow you to familiarize yourself with the Poisson distribution. The empirical continuous distribution works in a similar manner as the DISC() function except that the sample values are linearly interpolated. This process is discussed in the Arena™ help system and a general presentation is given in Banks et al. [2005] or Law [2007]. A discussion of modeling with the Johnson distribution can also be found in Law [2007].

## 6.10 SUMMARY

In this chapter, you learned how to analyze data in order to model the input distributions for a simulation model. The input distributions drive the stochastic behavior of the simulation model.

The modeling of the input distributions requires the following:

- Understanding how the system being modeled works. This understanding improves overall model construction in an iterative manner: model the system, observe some data, model the data, model the system, etc.
- Carefully collecting the data using well thought out collection and sampling plans.
- Analyzing the data using appropriate statistical techniques.
- Hypothesizing and testing appropriate probability distributions.
- Incorporating the models into your simulations.

Properly modeling the inputs to the simulation model form a critical foundation to increase the validity of the simulation model and, subsequently, the credibility of the simulation outputs. Chapter 7 begins our discussion of how to properly analyze the stochastic output from a discrete-event simulation model.

## EXERCISES

- 6.1 The observations available in the text file *ch6problem1.txt* represent the count of the number of failures on a windmill turbine farm per year. Using the techniques discussed in the chapter, recommend an input distribution model for this situation.
- 6.2 The observations available in the text file *ch6problem2.txt* represent the time that it takes to repair a windmill turbine on each occurrence in minutes. Using the techniques discussed in the chapter, recommend an input distribution model for this situation.
- 6.3 The observations available in the text file *ch6problem3.txt* represent the time in minutes that a repair person takes to drive to the windmill farm to repair a failed turbine. Using the techniques discussed in the chapter, recommend an input distribution model for this situation.
- 6.4 The observations available in the text file *ch6problem4.txt* represent the time in seconds that it takes to service a customer at a movie theater counter. Using the techniques discussed in the chapter, recommend an input distribution model for this situation.
- 6.5 The observations available in the text file *ch6problem5.txt* represent the time in hours between failures of a critical piece of computer testing equipment. Using the techniques discussed in the chapter, recommend an input distribution model for this situation.
- 6.6 The observations available in the text file *ch6problem6.txt* represent the time in minutes associated with performing a lube, oil, and maintenance check at the local Quick Oil Change Shop. Using the techniques discussed in the chapter, recommend an input distribution model for this situation.
- 6.7 If  $Z \sim N(0, 1)$  and  $Y = \sum_{i=1}^k Z_i^2$ , then  $Y \sim \chi_k^2$ , where  $\chi_k^2$  is a chi-squared random variable with  $k$  degrees of freedom. Set up an Arena™ model to generate  $n = 32, 64, 128, 256, 1024$  observations of  $Y$  with  $k = 5$ . For each sample, fit a distribution to the sample.
- 6.8 Consider the following sample that represents the time (in seconds) a hematology cell counter takes to complete a test on a blood sample.

23.79	75.51	29.89	2.47	32.37
29.72	84.69	45.66	61.46	67.23
94.96	22.68	86.99	90.84	56.49
30.45	69.64	17.09	33.87	98.04
12.46	8.42	65.57	96.72	33.56
35.25	80.75	94.62	95.83	38.07
14.89	54.80	95.37	93.76	83.64
50.95	40.47	90.58	37.95	62.42
51.95	65.45	11.17	32.58	85.89
65.36	34.27	66.53	78.64	58.24

Test the hypothesis that these data are drawn from a uniform distribution at a 95% confidence level, given the following information:

- The interval of the distribution is between 0 and 100.
- The interval of the distribution is between  $a$  and  $b$ , where  $a$  and  $b$  are unknown parameters.

- 6.9 Consider the following output for fitting a uniform distribution to a data set with the Arena Input Analyzer. Would you reject or not reject the hypothesis that the data is uniformly distributed.

```

        Distribution Summary
Distribution: Uniform
Expression: UNIF(36, 783)
Square Error: 0.156400

Chi Square Test
Number of intervals = 7
Degrees of freedom = 6
Test Statistic      = 164
Corresponding p-value < 0.005

Kolmogorov-Smirnov Test
Test Statistic = 0.495
Corresponding p-value < 0.01

        Data Summary
Number of Data Points = 100
Min Data Value       = 36.8
Max Data Value       = 782
Sample Mean          = 183
Sample Std Dev       = 142

        Histogram Summary
Histogram Range      = 36 to 783
Number of Intervals = 10

```

- 6.10 Consider the following frequency data on the number of orders received per day,  $x_j$ , by a warehouse.

$j$	$x_j$	$c_j$	$np_j$	$\frac{(c_j - np_j)^2}{np_j}$
1	0	10		
2	1	42		
3	2	27		
4	3	12		
5	4	6		
6	5 or more	3		
	Total	100		

- (a) Compute the sample mean for this data.
- (b) Perform a chi-squared goodness-of-fit test to test the hypothesis (use a 95% confidence level) that the data is Poisson distributed. Complete the provided table to show your calculations.
- 6.11 The number of electrical outlets in a prefabricated house varies between 10 and 22 outlets. Since the time to perform the install depends on the number of outlets, data was collected to develop a probability distribution for this variable. The data set is

given below:

14	16	14	16	12	14	12	13	12	12
12	12	14	22	13	16	15	15	15	21
18	12	17	14	13	11	12	14	10	10
16	12	13	11	13	11	11	12	13	16
15	12	11	14	11	12	11	11	13	17

Fit a probability model to the number of electrical outlets per prefabricated house.

- 6.12 Generate 100 uniform(0,1) numbers using Arena<sup>TM</sup>. Use a CREATE module to create 100 entities. Use an ASSIGN module and the expression builder to assign the randomly generated number to a variable. Use a WRITE module to write the value of the variable to a text file. Use a DISPOSE module to dispose of the entities.
- (a) Generate 100 uniform(0,1) numbers using Arena<sup>TM</sup>.
  - (b) Use the K-S test with alpha = 0.05 to test if the hypothesis that the numbers are uniformly distributed on the (0,1) interval can be rejected.
  - (c) Use a chi-squared goodness-of-fit test with 10 intervals and alpha = 0.05 to test if the hypothesis that the numbers are uniformly distributed on the (0,1) interval can be rejected.
- 6.13 Test the following fact by generating instances of  $Y = \sum_{i=1}^r X_i$ , where  $X_i \sim \text{expo}(\beta)$  and  $\beta = E[X_i]$ . Use  $r = 5$  and  $\beta = 2$ . Be careful to specify the correct parameters for the exponential distribution function in Arena, for example, the exponential distribution takes in the mean of the distribution as its parameter. Generate 10, 100, 1000 instances of  $Y$ .
- (a) Perform hypothesis tests to check  $H_0 : \mu = \mu_0$  versus  $H_1 : \mu \neq \mu_0$  where  $\mu$  is the true mean of  $Y$  for each of the sample sizes generated.
  - (b) Use the same samples to fit a distribution using the Input Analyzer. Properly interpret the statistical results supplied by the Input Analyzer.
- 6.14 Suppose that we are interested in modeling the arrivals to Sly's BBQ Food Truck during 11:30 am to 1:30 pm in the downtown square. During this time, a team of undergraduate students has collected the number of customers arriving to the truck for 10 different periods of the 2-hour time frame for each day of the week. The data is given below:

Obs#	Day of Week						
	M	T	W	R	F	S	SU
1	13	4	4	3	8	8	9
2	6	5	7	7	5	6	8
3	7	14	10	5	5	5	10
4	12	6	10	5	12	7	4
5	6	8	8	5	4	11	9
6	10	6	9	3	3	6	4
7	9	5	5	5	7	5	4
8	7	10	11	9	7	10	13
9	8	4	2	7	6	5	7
10	9	2	6	8	7	4	9

- (a) Visualize the data.
  - (b) Check if the day of the week influences the statistical properties of the count data.
  - (c) Tabulate the frequency of the count data.
  - (d) Estimate the mean rate parameter of the hypothesized Poisson distribution.
  - (e) Perform goodness-of-fit tests to test the hypothesis that the number of arrivals for the interval 11:30 am to 1:30 pm has a Poisson distribution versus the alternative that it does not have a Poisson distribution.
- 6.15 Use Arena™ to generate 100 observations of the triangular distribution with  $a = 2$ ,  $c = 5$ ,  $b = 10$ . Using the techniques discussed in the chapter, test the hypothesis that the observations come from a triangular distribution with parameters  $a = 2$ ,  $c = 5$ ,  $b = 10$ .
- 6.16 A copy center has one fast copier and one slow copier. The copy time per page for the fast copier is thought to be lognormally distributed with a mean of 1.6 seconds and a standard deviation of 0.3 seconds. A co-op Industrial Engineering student has collected some time study data on the time to copy a page for the slow copier. The times, in seconds, are given in the following table:

3.05	0.56	2.54	1.59	0.9	4.02	2.01	1.85	1.76	7.75
0.99	1.69	6.53	2.02	3.19	3.03	2.21	1.01	3.44	0.26
2.86	4.03	3.13	3.22	4.16	3.53	2.14	1.58	2.44	0.43
3.01	1.94	0.62	0.98	4.12	3.38	3.74	3.63	4.87	4.45
3.54	2.12	7.51	3.47	8.41	1.35	1.23	2.47	5.25	4.2
1.87	1.67	2.35	4.59	6.55	1.22	1.23	0.72	1.69	1.68
3.66	2.04	3.05	2.47	1.39	5.28	4.51	2.84	0.21	3.41
1.16	4.39	2.65	2.15	2.56	1.13	2.75	2.06	0.84	0.69
1.79	2.86	6.06	1.38	4.67	2.29	2.65	1.02	3.11	2.76
1.26	2.42	4.49	1.28	4.16	2.44	7.16	3.98	4.06	5.08

The copy times for the slow and fast copiers are given on a per page basis. Thus, the total time to perform a copy job of  $N$  pages is the sum of the copy times for the  $N$  individual pages. Each individual page's time is random.

Customers arrive to the copy center according to a Poisson process with a mean rate of one customer every 40 seconds. The number of copies requested by each customer is equally likely over the range of 10–50 copies. The customer is responsible for filling out a form that indicates the number of copies to be made. This results in a copy job which is processed by the copying machines in the copy center. The copying machines work on the entire job at one time.

The policy for selecting a copier is as follows: if the number of copies requested is less than or equal to 30, the slow copier will be used. If the number of copies exceeds 30, the fast copier will be used, with one exception: if no jobs are in progress on the slow copier and the number of jobs waiting for the fast copier is at least two, then the customer will be served by the slow copier. After the customer gives the originals for copying, the customer proceeds to the service counter to pay for the copying. Assume that giving the originals for copying requires no time and thus does not require action by the copy center personnel. In addition, assume that one cashier handles the payment counter only so that sufficient workers are available to run the copy machines. The

time to complete the payment transaction is lognormally distributed with a mean of 20 seconds and a standard deviation of 10 seconds. As soon as both the payment and the copying job are finished, the customer takes the copies and departs the copying center. The copy center starts out a day with no customers and is open for 10 hours per day.

Management has requested that the co-op Industrial Engineer develops a model because they are concerned that customers have to wait too long for copies. Recently, several customers complained about long waits. Their standard is that the probability that a customer waits longer than 4 minutes should be no more than 10%. They define a customer's waiting time as the time interval from when the customer enters the store to the time the customer leaves the store with their completed copy job. If the waiting time criteria is not met, several options are available: The policy for allocating jobs to the fast copier could be modified or the company could purchase an additional copier which could be either a slow copier or a fast copier. Develop an Arena™ model for this problem. Based on 25 replications, report in table form, the appropriate statistics on the waiting time of customers, the daily throughput of the copy center, and the utilization of the payment clerk. In addition, estimate the probability that a customer waits longer than 4 minutes.



---

# 7

---

## ANALYZING SIMULATION OUTPUT

### LEARNING OBJECTIVES

- To be able to recognize the different types of statistical quantities used within and produced by simulation models.
- To be able to analyze finite-horizon simulations via the method of replications.
- To be able to analyze infinite-horizon simulations via the method of batch means and the method of replication–deletion.
- To be able to compare simulation alternatives and make valid decisions based on the statistical output of a simulation.
- To be able to model systems with shared resources and apply the Arena<sup>TM</sup> constructs involved in using resource sets.

Chapter 6 explored the issue of randomness within the inputs to a simulation model. Because the inputs to the simulation are random, the outputs from the simulation are also random. This concept is illustrated in Figure 7.1. You can think of a simulation model as a function that maps inputs to outputs. This chapter presents the statistical analysis of the outputs from simulation models. This builds on the material first presented in Section 3.4.

In addition, a number of issues that are related to the proper execution of simulation experiments are presented. For example, the simulation outputs are dependent upon the input random variables, input parameters, and the initial conditions of the model. Initial conditions refer to the starting conditions for the model, that is, whether or not the system starts “empty and idle.” The effect of initial conditions on steady-state simulations is discussed in this chapter.



**Figure 7.1** Random input implies random output.

Input parameters are related to the controllable and uncontrollable factors associated with the system. For a simulation model, *all* input parameters are controllable; however, in the system being modeled, you typically have control over only a limited set of parameters. Thus, in simulation, you have the unique ability to control the random inputs into your model. This chapter discusses how to take advantage of controlling the random inputs.

Input parameters can be further classified as decision variables. That is, those parameters of interest that you want to change in order to test model configurations for decision making. The structure of the model itself may be considered a decision variable when you are trying to optimize the performance of the system. When you change the input parameters for the simulation model and then execute the simulation, you are simulating a different design alternative.

This chapter describes how to analyze the output from a single design alternative and how to analyze the results of multiple design alternatives. To begin the discussion, you need to build an understanding of the types of statistical quantities that may be produced by a simulation experiment.

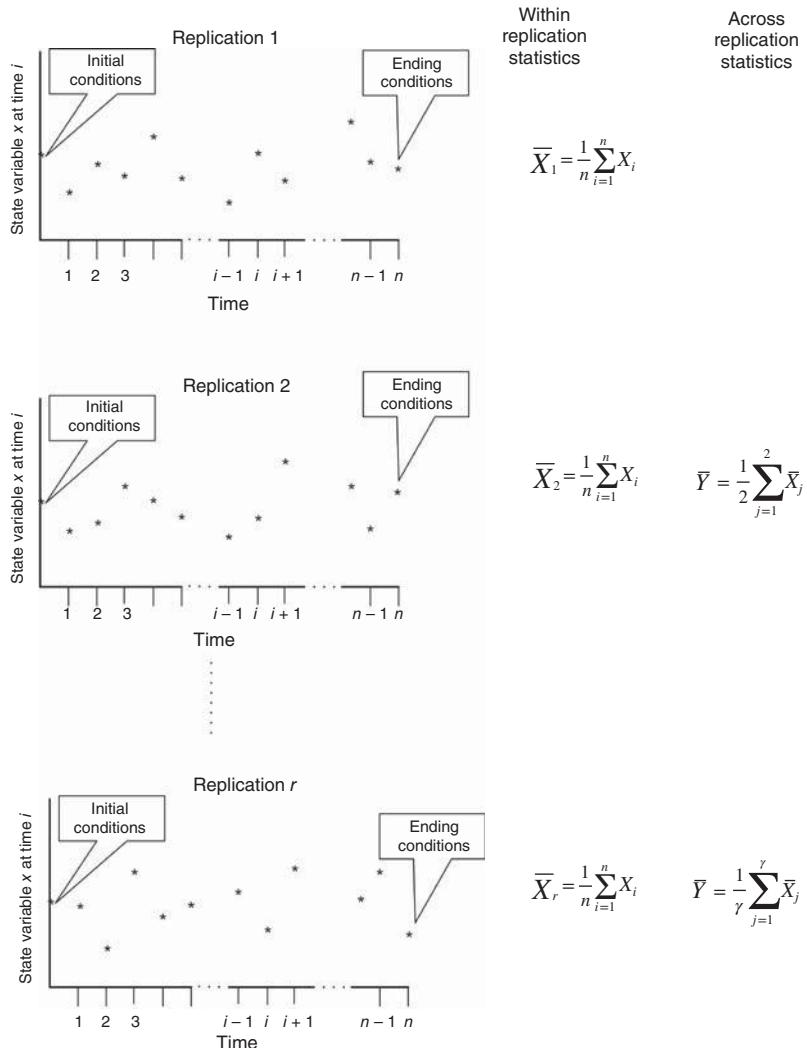
## 7.1 TYPES OF STATISTICAL VARIABLES

A simulation experiment occurs when the modeler sets the input parameters to the model and executes the simulation. This causes events to occur and the simulation model to evolve over time. During the execution of the simulation, the behavior of the system is observed and various statistical quantities computed. When the simulation reaches its termination point, the statistical quantities are summarized in the form of output reports.

A simulation experiment may be for a single replication of the model or may have multiple replications. A *replication* is the generation of one sample path which represents the evolution of the system from its initial conditions to its ending conditions. If you have multiple replications within an experiment, each replication represents a different sample path, starting from the same initial conditions and being driven by the same input parameter settings. Because the randomness within the simulation can be controlled, the underlying random numbers used within each replication of the simulation can be made to be independent. Thus, as the name implies, each replication is an independently generated “repeat” of the simulation. Figure 7.2 illustrates the concept of replications being repeated (independent) sample paths.

Within a single sample path (replication), the statistical behavior of the model can be observed.

**Definition 7.1 (Within-Replication Statistics)** The statistical quantities collected during a replication are called *within-replication statistics*.



**Figure 7.2** The concept of replicated sample paths.

**Definition 7.2 (Across Replication Statistics)** The statistical quantities collected across replications are called *across replication statistics*. Across replication statistics are collected based on the observation of the final values of within replication statistics.

*Within-replication statistics* are collected based on the observation of the sample path and include observations on entities, state changes, etc. that occur during a sample path execution. The observations used to form within-replication statistics are not likely to be independent and identically distributed. Since across replication statistics are formed from the final values of within-replication statistics, one observation per replication is available. Since each replication is considered independent, the observations that form the sample for across replication statistics are likely to be independent and identically distributed. The statistical properties of within and across replication statistics are inherently different and

require different methods of analysis. Of the two, within-replication statistics are the more challenging from a statistical standpoint.

For within-replication statistical collection, there are two primary types of observations: *tally* and *time persistent*. Tally observations represent a sequence of equally weighted data values that do not persist over time. This type of observation is associated with the duration or interval of time that an object is in a particular state or how often the object is in a particular state. As such it is observed by marking (tallying) the time that the object enters the state and the time that the object exits the state. Once the state change takes place, the observation is over (it is gone, it does not persist, etc.). If we did not observe the state change, then we would have missed the observation. The time spent in queue, the count of the number of customers served, whether or not a particular customer waited longer than 10 minutes are all examples of tally observations.

Time-persistent observations represent a sequence of values that persist over some specified amount of time with that value being weighted by the amount of time over which the value persists. These observations are directly associated with the values of the state variables within the model. The value of a time-persistent observation persists in time. For example, the number of customers in the system is a common state variable. If we want to collect the average number of customers in the system *over time*, then this will be a time-persistent statistic. While the value of the number of customers in the system changes at discrete points in time, it holds (or persists with) that value over a duration of time. This is why it is called a time-persistent variable.

Figure 7.3 illustrates a single sample path for the number of customers in a queue over a period of time. From this sample path, events and subsequent statistical quantities can be observed.

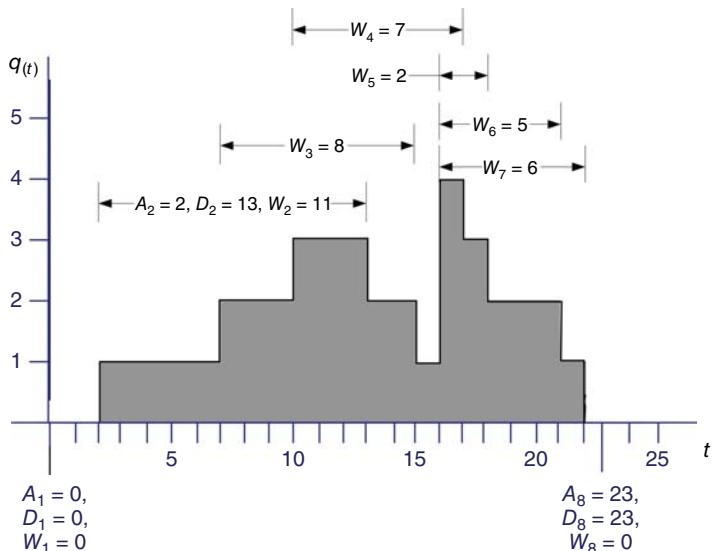


Figure 7.3 Sample path for tally and time-persistent data.

- Let  $A_i$   $i = 1, \dots, n$  represent the time that the  $i$ th customer enters the queue.
- Let  $D_i$   $i = 1, \dots, n$  represent the time that the  $i$ th customer exits the queue.
- Let  $W_i = D_i - A_i$   $i = 1, \dots, n$  represent the time that the  $i$ th customer spends in the queue.

Thus,  $W_i$   $i = 1, \dots, n$  represents the sequence of wait times for the queue, each of which can be individually observed and tallied. This is tally type data because the customer enters a state (the queued state) at time  $A_i$  and exits the state at time  $D_i$ . When the customer exits the queue at time  $D_i$ , the waiting time in queue  $W_i = D_i - A_i$  can be observed or tallied.  $W_i$  is only observable at the instant  $D_i$ . This makes  $W_i$  tally-based data and, once observed, its value never changes again with respect to time. Tally data is most often associated with an entity that is moving through states that are implied by the simulation model. An observation becomes available each time the entity enters and subsequently exits the state.

With tally data, it is natural to compute the sample average as a measure of the central tendency of the data. Assume that you can observe  $n$  customers entering and existing the queue, then the average waiting time across the  $n$  customers is given by

$$\bar{W}(n) = \frac{1}{n} \sum_{i=1}^n W_i$$

Many other statistical quantities, such as the minimum, maximum, and sample variance, can also be computed from these observations. Unfortunately, within-replication data is often (if not always) correlated with respect to time. In other words, within-replication observations like  $W_i$   $i = 1, \dots, n$  are not statistically independent. In fact, they are likely to also not be identically distributed. Both of these issues will be discussed when the analysis of infinite-horizon or steady-state simulation models is presented.

The other type of statistical variable encountered within a replication is based on time-persistent observations. Let  $q(t)$ ,  $t_0 < t \leq t_n$ , be the number of customers in the queue at time  $t$ . Note that  $q(t) \in \{0, 1, 2, \dots\}$ . As illustrated in Figure 7.3,  $q(t)$  is a function of time (a step function in this particular case). That is, for a given (realized) sample path,  $q(t)$  is a function that returns the number of customers in the queue at time  $t$ .

The mean value theorem of calculus for integrals states that given a function,  $f(\cdot)$ , continuous on an interval  $[a, b]$ , there exists a constant,  $c$ , such that

$$\int_a^b f(x) dx = f(c)(b - a) \quad (7.1)$$

The value  $f(c)$  is called the mean value of the function. A similar function can be defined for  $q(t)$ . In simulation, this function is called the time average:

$$\bar{L}_q(n) = \frac{1}{t_n - t_0} \int_{t_0}^{t_n} q(t) dt \quad (7.2)$$

This function represents the average with respect to time of the given state variable. This type of statistical variable is called time persistent because  $q(t)$  is a function of time (i.e., it persists over time).

In the particular case where  $q(t)$  represents the number of customers in the queue,  $q(t)$  will take on constant values during intervals of time corresponding to when the queue has a certain number of customers. Let  $q(t) = q_k$  for  $t_{k-1} \leq t \leq t_k$  and define  $v_k = t_k - t_{k-1}$ , then the time-average number in queue can be rewritten as follows:

$$\begin{aligned}\bar{L}_q(n) &= \frac{1}{t_n - t_0} \int_{t_0}^{t_n} q(t) dt \\ &= \frac{1}{t_n - t_0} \sum_{k=1}^n q_k (t_k - t_{k-1}) \\ &= \frac{\sum_{k=1}^n q_k v_k}{t_n - t_0} \\ &= \frac{\sum_{k=1}^n q_k v_k}{\sum_{k=1}^n v_k}\end{aligned}$$

Note that  $q_k(t_k - t_{k-1})$  is the area under  $q(t)$  over the interval  $t_{k-1} \leq t \leq t_k$  and

$$t_n - t_0 = \sum_{k=1}^n v_k = (t_1 - t_0) + (t_2 - t_1) + \cdots + (t_{n-1} - t_{n-2}) + (t_n - t_{n-1})$$

is the total time over which the variable is observed. Thus, the time average is simply the area under the curve divided by the amount of time over which the curve is observed. From this equation, it should be noted that each value of  $q_k$  is weighted by the length of time that the variable has the value. This is why the time average is often called the time-weighted average. If  $v_k = 1$ , then the time average is the same as the sample average.

With time-persistent data, you often want to estimate the percentage of time that the variable takes on a particular value. Let  $T_i$  denote the *total* time during  $t_0 < t \leq t_n$  that the queue had  $q(t) = i$  customers. To compute  $T_i$ , you sum all the rectangles corresponding to  $q(t) = i$  in the sample path. Because  $q(t) \in \{0, 1, 2, \dots\}$ , there are an infinite number of possible value for  $q(t)$  in this example; however, within a finite sample path, you can only observe a finite number of the possible values. The ratio of  $T_i$  to  $T = t_n - t_0$  can be used to estimate the percentage of time the queue had  $i$  customers. That is, define  $\hat{p}_i = T_i/T$  as an estimate of the proportion of time that the queue had  $i$  customers during the interval of observation.

### EXAMPLE 7.1 Queue Statistics

Consider Figure 7.3 which shows the variation in queue length over a simulated period of 25 time units.

1. Compute the time-average number in queue over the interval of time from 0 to 25.
2. Compute the percentage of time that the queue had  $\{0, 1, 2, 3, 4\}$  customers.

*Solution to Example 7.1* Consider Figure 7.3 which shows the variation in queue length over a simulated period of 25 time units. Since the queue length is a time-persistent variable,

the time-average queue length can be computed as

$$\begin{aligned}\bar{L}_q &= \frac{0(2-0) + 1(7-2) + 2(10-7) + 3(13-10) + 2(15-13) + 1(16-15)}{25} \\ &\quad + \frac{4(17-16) + 3(18-17) + 2(21-18) + 1(22-21) + 0(25-22)}{25} \\ &= \frac{39}{25} = 1.56\end{aligned}$$

To estimate the percentage of time that the queue had  $\{0, 1, 2, 3, 4\}$  customers, the values of  $v_k = t_k - t_{k-1}$  need to be summed for whenever  $q(t) \in \{0, 1, 2, 3, 4\}$ . This results in the following:

$$\begin{aligned}\hat{p}_0 &= \frac{T_0}{T} = \frac{(2-0)+(25-22)}{25} = \frac{2+3}{25} = \frac{5}{25} = 0.2 \\ \hat{p}_1 &= \frac{T_1}{T} = \frac{(7-2)+(16-15)+(22-21)}{25} = \frac{5+1+1}{25} = \frac{7}{25} = 0.28 \\ \hat{p}_2 &= \frac{T_2}{T} = \frac{(3+2+3)}{25} = \frac{8}{25} = 0.28 \\ \hat{p}_3 &= \frac{T_3}{T} = \frac{3+1}{25} = \frac{4}{25} = 0.16 \\ \hat{p}_4 &= \frac{T_4}{T} = \frac{1}{25} = 0.04\end{aligned}$$

Notice that the sum of the  $\hat{p}_i$  adds to 1. To compute the average waiting time in the queue, use the supplied values for each waiting time.

$$\bar{W}(8) = \frac{\sum_{i=1}^n W_i}{n} = \frac{0+11+8+7+2+5+6+0}{8} = \frac{39}{8} = 4.875$$

Notice that there were two customers, one at time 1.0 and another at time 23.0 that had waiting times of zero. The state graph did not move up or down at those times. Each unit increment in the queue length is equivalent to a new customer entering (and staying in) the queue. On the other hand, each unit decrement of the queue length signifies a departure of a customer from the queue. If you assume a first-in, first-out (FIFO) queue discipline, the waiting times of the six customers who entered the queue (and had to wait) are shown in the figure.

Now that we understand the type of data that occurs within a replication, we need to develop an understanding for the types of simulation situations that require specialized statistical analysis. The next section introduces this important topic.

## 7.2 TYPES OF SIMULATION WITH RESPECT TO OUTPUT ANALYSIS

When modeling a system, specific measurement goals for the simulation responses are often required. The goals, coupled with how the system operates, will determine how you execute

and analyze the simulation experiments. In planning the experimental analysis, it is useful to think of simulations as consisting of two main categories related to the period of time over which a decision needs to be made.

**Finite Horizon** In a finite-horizon simulation, a well-defined ending time or ending condition can be specified, which clearly defines the end of the simulation. Finite-horizon simulations are often called *terminating* simulations, since there are clear terminating conditions.

**Infinite Horizon** In an infinite-horizon simulation, there is no well-defined ending time or condition. The planning period is over the life of the system, which from a conceptual standpoint lasts forever. Infinite-horizon simulations are often called *steady-state* simulations because in an infinite-horizon simulation, you are often interested in the long-term or steady-state behavior of the system.

For a finite-horizon simulation, an event or a condition associated with the system is present, which indicates the end of each simulation replication. This event can be specified in advance or its time of occurrence can be a random variable. If it is specified in advance, it is often because you do not want information past that point in time (e.g., a 3-month planning horizon). It might be a random variable in the case of the system stopping when a condition is met. For example, an ending condition may be specified to stop the simulation when there are no entities left to process. This was the case in the LOTR, Inc. example of Chapter 6. Finite-horizon simulations are very common since most planning processes are finite. A few example systems involving a finite horizon include

- Bank: bank doors open at 9 am and close at 5 pm
- Military battle: simulate until force strength reaches a critical value
- Filling a customer order: suppose a new contract is accepted to produce 100 products, you might simulate the production of the 100 products to see the cost, delivery time, etc. The LOTR Ring Makers example of Chapter 6 is an example of this situation.

For a finite-horizon simulation, each replication represents a sample path of the model for one instance of the finite horizon. The length of the replication corresponds to the finite horizon of interest. For example, in modeling a bank that opens at 9 am and closes at 5 pm, the length of the replication would be 8 hours.

In contrast to a finite-horizon simulation, an infinite-horizon simulation has no natural ending point. Of course, when you actually simulate an infinite-horizon situation, a finite replication length must be specified. Hopefully, the replication length will be long enough to satisfy the goal of observing long-run performance. Examples of infinite-horizon simulations include

- A factory where you are interested in measuring the steady-state throughput
- A hospital emergency room which is open 24 hours a day, 7 days of week
- A telecommunications system which is always operational.

Infinite-horizon simulations are often tied to systems that operate continuously and for which the long-run or steady-state behavior needs to be estimated.

Because infinite-horizon simulations often model situations where the system is always operational, they often involve the modeling of non-stationary processes. In such situations,

care must be taken in defining what is meant by long-run or steady-state behavior. For example, in an emergency room that is open 24 hours a day, 365 days per year, the arrival pattern to such a system probably depends on time. Thus, the output associated with the system is also non-stationary. The concept of steady state implies that the system has been running so long that the system's behavior (in the form of performance measures) no longer depends on time; however, in the case of the emergency room, since the inputs depend on time so do the outputs. In such cases, it is often possible to find a period of time or cycle over which the non-stationary behavior repeats. For example, the arrival pattern to the emergency room may depend on the day of the week, such that every Monday has the same characteristics, every Tuesday has the same characteristics, and so on for each day of the week. Thus, on a weekly basis, the non-stationary behavior repeats. You can then define your performance measure of interest based on the appropriate non-stationary cycle of the system. For example, you can define  $Y$  as the expected waiting time of patients *per week*. This random variable may have performance that can be described as long term. In others, the long-run weekly performance of the system may be stationary. This type of simulation has been termed steady-state cyclical parameter estimation within Law (2007).

Of the two types of simulations, finite-horizon simulations are easier to analyze. Luckily, they are the type of simulation typically found in practice. In fact, when you think that you are faced with an infinite-horizon simulation, you should very carefully evaluate the goals of your study to see if they can just as well be met with a finite planning horizon. The analysis of both of these types of simulations are discussed in this chapter through examples.

### 7.3 ANALYSIS OF FINITE-HORIZON SIMULATIONS

This section illustrates how tally-based and time-persistent statistics are collected within a replication and how statistics are collected across replications. Finite-horizon simulations can be analyzed by traditional statistical methodologies that assume a random sample, that is, independent and identically distributed random variables. A simulation experiment is the collection of experimental design points (specific input parameter values) over which the behavior of the model is observed. For a particular design point, you may want to repeat the execution of the simulation multiple times to form a sample at that design point. To get a random sample, you execute the simulation starting from the same initial conditions and ensure that the random numbers used within each replication are independent. Each replication must also be terminated by the same conditions. It is very important to understand that independence is achieved across replications, that is, the replications are independent. The data *within* a replication may or may not be independent.

The method of *independent replications* is used to analyze finite-horizon simulations. Suppose that  $n$  replications of a simulation are available where each replication is terminated by some event  $E$  and begun with the same initial conditions. Let  $Y_{rj}$  be the  $j$ th observation on replication  $r$  for  $j = 1, 2, \dots, m_r$ , where  $m_r$  is the number of observations in the  $r$ th replication, and  $r = 1, 2, \dots, n$ , and define the sample average for each replication to be

$$\bar{Y}_r = \frac{1}{m_r} \sum_{j=1}^{m_r} Y_{rj} \quad (7.3)$$

If the data are time based, then

$$\bar{Y}_r = \frac{1}{T_E} \int_0^{T_E} Y_r(t) dt \quad (7.4)$$

$\bar{Y}_r$  is the sample average based on the observation within the  $r$ th replication. It is a random variable that can be observed at the end of each replication; therefore,  $\bar{Y}_r$  for  $r = 1, 2, \dots, n$  forms a random sample. Thus, standard statistical analysis of the random sample can be performed.

To make this concrete, suppose that you are examining a bank that opens with no customers at 9 am and closes its doors at 5 pm to prevent further customers from entering. Let  $W_{rj}$ ,  $j = 1, \dots, m_r$ , represent the sequence of waiting times for the customers that entered the bank between 9 am and 5 pm on day (replication)  $r$  where  $m_r$  is the number of customers who were served between 9 am and 5 pm on day  $r$ . For simplicity, ignore the customers who entered before 5 pm but did not get served until after 5 pm. Let  $N_r(t)$  be the number of customers in the system at time  $t$  for day (replication)  $r$ . Suppose that you are interested in the mean daily customer waiting time and the mean number of customers in the bank on any given 9 am to 5 pm day, that is, you are interested in  $E[W_r]$  and  $E[N_r]$  for any given day. At the end of each replication, the following can be computed:

$$\begin{aligned}\bar{W}_r &= \frac{1}{m_r} \sum_{j=1}^{m_r} W_{rj} \\ \bar{N}_r &= \frac{1}{8} \int_0^8 N_r(t) dt\end{aligned}$$

At the end of all replications, random samples  $\bar{W}_1, \bar{W}_2, \dots, \bar{W}_n$  and  $\bar{N}_1, \bar{N}_2, \dots, \bar{N}_n$  are available from which sample averages, standard deviations, confidence intervals, etc. can be computed. Both of these samples are based on observations of within-replication data.

Both  $\bar{W}_r$  and  $\bar{N}_r$  are averages of many observations within the replication. Sometimes, there may only be one observation based on the entire replication. For example, suppose that you are interested in the probability that someone is still in the bank when the doors close at 5 pm, that is, you are interested in  $\theta = P\{N(t = 5 \text{ pm}) > 0\}$ . In order to estimate this probability, an indicator variable can be defined within the simulation and observed each time the condition was met or not. For this situation, an indicator variable,  $I_r$ , for each replication can be defined as follows:

$$I_r = \begin{cases} 1 & N(t = 5 \text{ pm}) > 0 \\ 0 & N(t = 5 \text{ pm}) \leq 0 \end{cases}$$

Therefore, at the end of the replication, the simulation must tabulate whether or not there are customers in the bank and record the value of this indicator variable. Since this happens only once per replication, a random sample of the  $I_1, I_2, \dots, I_n$  will be available after all replications have been executed. As we learned in Chapter 3, we can use the observations of the indicator variable to estimate the desired probability.

Since the analysis of the system will be based on a random sample, the key design criteria for the experiment will be the required number of replications. In other words, you need to determine the sample size.

Because confidence intervals may form the basis for decision making, you can use the confidence interval half-width in determining the sample size. For example, in estimating  $E[W_r]$  for the bank example, you might want to be 95% confident that you have estimated the true waiting time to within  $\pm 2$  minutes.

There are three related methods that are commonly used for determining the sample size for this situation:

- an iterative method based on the  $t$ -distribution,
- an approximate method based on the normal distribution, and
- the half-width ratio method.

Each of these methods assumes that the observations in the sample are independent and identically distributed from a normal distribution. In addition, the methods also assume that the pilot replications are representative of the population under study. When the data are not normally distributed, you must rely on the central limit theorem to get approximate results. The assumption of normality is typically justified when across replication statistics are based on within-replication averages.

Chapter 3 presented the first two methods. This chapter discusses the half-width ratio method because it is the easiest to utilize based on Arena<sup>TM</sup> output reports.

### 7.3.1 Determining the Number of Replications

When multiple replications are executed in Arena<sup>TM</sup>, the output reports automatically provide a 95% confidence interval for the performance measures. Arena<sup>TM</sup> does not report the standard deviation, but instead directly reports the half-width value for a 95% confidence interval. If you make a pilot run of  $n_0$  replications, you can use the reported half-width to determine how many replications you need to have to be close to a desired half-width bound. To do this, you can use the half-width ratio method.

Let  $h_0$  be the initial value for the half-width from the pilot run of  $n_0$  replications.

$$h_0 = t_{\alpha/2, n_0-1} \frac{s_0}{\sqrt{n_0}} \quad (7.5)$$

Solving for  $n_0$  yields

$$n_0 = t_{\alpha/2, n_0-1}^2 \frac{s_0^2}{h_0^2} \quad (7.6)$$

Similarly, for any  $n$ , we have

$$n = t_{\alpha/2, n-1}^2 \frac{s^2}{h^2} \quad (7.7)$$

Taking the ratio of  $n_0$  to  $n$  (Equations 7.6 and 7.7) and assuming that  $t_{\alpha/2, n-1}$  is approximately equal to  $t_{\alpha/2, n_0-1}$  and  $s^2$  is approximately equal to  $s_0^2$  yields

$$n \cong n_0 \frac{h_0^2}{h^2} = n_0 \left( \frac{h_0}{h} \right)^2 \quad (7.8)$$

This is the half-width ratio equation. Figure 7.4 illustrates the spreadsheet in *SampleSizeDetermination.xls* that facilitates this calculation.

	A	B	C	D	E
1					
2					
3	Initial Half-Width	47.27			
4	Initial Number of Replications	10			
5	Desired Half-Width	20			
6	Required Sample Size	55.86132			
7					
8					
9					
10					
11					
12					

Arena gives you the half-width for a 95% confidence interval when performing replications.  
 1) Plug in half-width from Arena output into B3  
 2) Plug in number of initial replications in B4  
 3) Plug in desired half-width into B6

Figure 7.4 Half-width ratio method within a spreadsheet.

In the case of an indicator variable such as  $I_r$ , which was suggested for use in estimating the probability that there are customers in the bank after 5 pm, the sampled observations are clearly not normally distributed. In this case, since you are estimating a proportion, you can use the sample size determination techniques for estimating proportions described in Chapter 3.

### ■ EXAMPLE 7.2 Illustrating Sample Size Calculations

Suppose a pilot run of a simulation model estimated that the average waiting time for a customer during the day was 11.485 minutes based on an initial sample size of 15 replications with a 95% confidence interval half-width of 1.04. Using the three sample size determination techniques, recommend a sample size to be 95% confident that you are within  $\pm 0.10$  of the true mean waiting time in the queue.

*Solution to Example 7.2* First, we will do the half-width ratio method. We have that  $h_0 = 1.04$ ,  $n_0 = 15$ , and  $h = 0.1$ , thus

$$n \cong n_0 \left( \frac{h_0}{h} \right)^2 = 15 \left( \frac{1.04}{0.1} \right)^2 = 1622.4 \cong 1623$$

To estimate a sample size based on the normal approximation method, we need to have the estimate of the initial sample standard deviation. Unfortunately, this is not directly reported, but it can be computed using Equation (7.5). Rearranging Equation (7.6) to solve for  $s_0$  yields

$$s_0 = \frac{h_0 \sqrt{n_0}}{t_{\alpha/2, n_0 - 1}} \quad (7.9)$$

Since we have a 95% confidence interval with  $n_0 = 15$ , we have that  $t_{0.025, 14} = 2.145$ , which yields

$$s_0 = \frac{h_0 \sqrt{n_0}}{t_{\alpha/2, n_0 - 1}} = \frac{1.04 \sqrt{15}}{2.145} = 1.87781$$

	B3				
	A	B	C	D	E
1					
2	alpha	0.05			
3	S	1.87781			
4	bound	0.1			
5	n	1348.799			
6	alpha/2	0.025			
7	t-alpha/2	1.961727			
8	half-width	0.100304			
9	difference	0.000304			
10					
11					
12					
13					
14					
15					
16					

Sample size determination  
Run goal seek to find the values of n such that the half-width is less than the specified desired bound.  
1) Specify alpha  
2) Specify S (standard deviation)  
3) Specify bound  
4) Specify initial n  
5) Run Goal Seek  
a) Tools> Goal Seek  
b) Set cell \$b\$9 to Value: 0 by changing cell \$b\$5

Figure 7.5 Iterative method for sample size via goal seek.

Now, we can use the normal approximation method. By using  $h$  as the desired bound  $E$  and  $z_{0.025} = 1.96$ , we have

$$n \geq \left( \frac{z_{\alpha/2} s}{E} \right)^2 = \left( \frac{1.96 \times 1.87781}{0.1} \right)^2 = 1354.54 \approx 1355$$

The final method is to use the iterative method based on the following equation:

$$h = t_{\alpha/2, n-1} \frac{s}{\sqrt{n}} \leq E$$

This can be accomplished within a spreadsheet as demonstrated in Chapter 3. Figure 7.5 presents the result of this analysis.

The three methods resulted in following recommendations:

- an iterative method based on the  $t$ -distribution,  $n = 1349$
- an approximate method based on the normal distribution,  $n = 1355$
- the half-width ratio method,  $n = 1623$

As noted in the discussion, the half-width ratio method recommended the largest number of replications.

### 7.3.2 Finite Horizon Example

This section reexamines the LOTR Ring Maker, Inc. model of Chapter 6 in order to form confidence intervals on the following additional quantities.

- The average number of pairs of rings in both the ring making process and the ring inspection process.

- The average time that it takes for a pair of rings to go through both the ring making process and the ring inspection process. In addition, a 95% confidence interval for the mean time to complete these processes to within  $\pm 20$  minutes is desired.

Arena™ can automatically report the average number of pairs of rings in the ring making process, the average number of pairs of rings in the inspection process, and the average time spent in each process separately. However, for this example, confidence intervals on the combined total are required, which is not automatically reported.

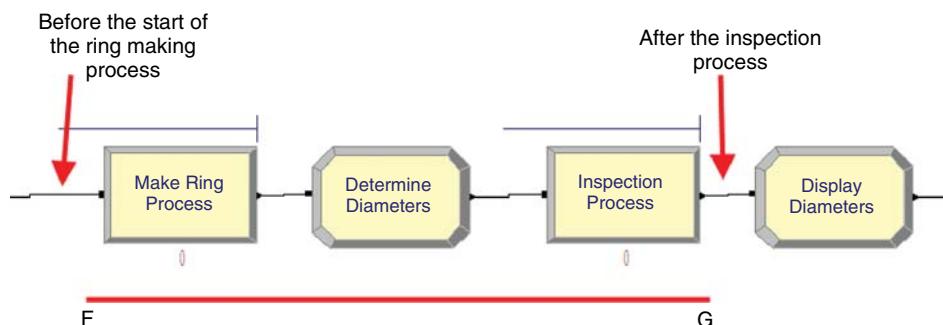
The following illustrates how to get the process reports automatically and how to add additional logic to collect the quantities of interest. The completed example is available in the file *LOTRCh7Example.doe*. If you are following along with the text, you can make a copy of the file *LOTRCh6Example.doe* and complete the steps outlined here. Open up a copy of the file *LOTRCh6Example.doe* and double-click on the Ring Processing sub-model. The start and end of the two processes that are involved in this example are denoted in Figure 7.6.

The pair of rings flows through each of these modules. The number of pairs of rings that are between the denoted points and how long that each pair of rings spends within the identified part of the flow process must be captured.

Let  $A_i$  denote the time that the  $i$ th pair of rings arrives to the Make Ring Process,  $D_i$  represent the time that the  $i$ th pair of rings departs from the Inspection Process, and  $N(t)$  be the number of pairs of rings between the points denoted as  $F$  and  $G$  within Figure 7.6.

The time spent within both processes for the  $i$ th pair of rings is  $D_i - A_i$ . Clearly,  $D_i - A_i$  is a tally-based quantity and  $N(t)$  is a time-persistent quantity. Within the model, you can use a variable to keep track of  $N(t)$ . Each time a pair of rings passes point  $F$ , the variable should be incremented, and each time a pair of rings passes point  $G$ , the variable should be decremented. Thus, the variable will keep track of the current number of entities (rings) in both processes. In the model, this variable is denoted as *vNumInMakeAndInspect*. You should use the VARIABLE data module to define this variable. Because the arrival time to the Make Ring Process is related to each entity, an attribute will be used to remember when the entity starts the ring making process. In the model, this attribute is denoted by *myArriveToMakeRingTime*.

There are two methods by which you can have time-persistent statistics collected on the variable *vNumInMakeAndInspect*. The first method can be accessed through the VARIABLE module, and the second method involves using the STATISTICS module on the Advanced Process panel. Both methods will be illustrated.



**Figure 7.6** Identifying the processes.

**(a)**

	Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics
1	vNumCalls			Real	System	1 rows	<input type="checkbox"/>
2	vSalesProb			Real	System	0 rows	<input type="checkbox"/>
3	vTol			Real	System	1 rows	<input type="checkbox"/>
4	vIDM			Real	System	1 rows	<input type="checkbox"/>
5	vIDS			Real	System	1 rows	<input type="checkbox"/>
6	vODM			Real	System	1 rows	<input type="checkbox"/>
7	vODS			Real	System	1 rows	<input type="checkbox"/>
8	vID			Real	System	0 rows	<input type="checkbox"/>
9	vOD			Real	System	0 rows	<input type="checkbox"/>
10	vGap			Real	System	0 rows	<input type="checkbox"/>
11	vNumInMakeAndInspect			Real	System	0 rows	<input checked="" type="checkbox"/>

Double-click here to add a new row.

**(b)**

Or define a Time-Persistent Statistic that uses the variable

**Statistic**

Name:  Type:

Expression:

Report Label:  Output File:

OK Cancel Help

Figure 7.7 Two ways to collect statistics on a time-persistent variable.

For the first method, select the VARIABLE data module and use the spreadsheet view to enter the name of the variable. Make sure that the Report Statistics option is checked as shown in Figure 7.7a. By checking the Report Statistics option, you are indicating that time-persistent statistics should be collected and reported on this variable during the simulation. This option is only available if the variable is a scalar (i.e., neither a 1D nor a 2D array variable).

The second option involves defining a user-defined time-persistent STATISTIC on the variable. You should do both ways to prove to yourself that they are the same. Go to the Advanced Process panel and select the STATISTIC data module. Double-click on the spreadsheet add row area to add another row and then right-click on the row. This will give the context menu to Edit Via Dialog. You should fill out the dialog box as shown in Figure 7.7b.

Use the name *AvgNumInMakeAndInspect* for the statistic, specify its type as time persistent and indicate that the expression to be used for collecting the statistics is simply the

variable name that is being used to track the number of entities in the Make and Inspect processes. This will create what is called a DSTAT variable to collect the time-based statistics for the given expression.

The advantage of using the STATISTIC module over the Report Statistics option within the VARIABLE module is that the STATISTIC module allows you to provide a different name for the reports and save the data to a file. When the quantity to be collected is more complicated than a simple variable, then the STATISTIC module's expression field should be used to define the appropriate expression. Thus, to collect time-persistent statistics on a quantity of interest in the simulation, you should use the time-persistent option within the STATISTIC module. Now, let us examine how to collect tally-based statistics.

To collect tally-based statistics, you need to place RECORD modules at appropriate locations in the model. In the case of collecting the quantity,  $D_i - A_i$ , you can first use an ASSIGN module to assign the value of  $A_i$  to *myArriveToMakeRingTime* and then use a RECORD module to collect the difference. As previously mentioned, the special variable TNOW will return the current simulation time. In this situation, you can use TNOW to get the values of  $A_i$  and  $D_i$  when the entity passes the appropriate points in the model. When the entity passes the point *F* in Figure 7.6, make the assignment

$$\text{myArriveToMakeRingTime} = \text{TNOW}$$

This is the value of  $A_i$  for the entity. Then, when the entity reaches point *G* of Figure 7.6, record the expression

$$\text{TNOW} - \text{myArriveToMakeRingTime}$$

This expression represents the elapsed *interval* of time that the entity took to move from point *F* to point *G* of Figure 7.6. Since this type of recording is so common, the RECORD module provides a special option to facilitate this operation.

The RECORD module has five different record types:

**Count** When Count is chosen, the module defines a COUNTER variable. This variable will be incremented or decremented by the value specified.

**Entity Statistics** This option will cause entity statistics to be recorded. Entity statistics will be discussed in Chapter 10.

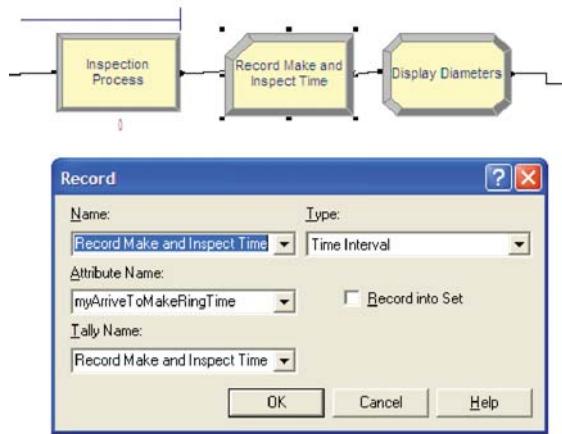
**Time Interval** When Time Interval is chosen, the module will take the difference between the current time, TNOW, and the specified attribute.

**Time Between** When Time Between is chosen, the module will take the difference between the current time, TNOW, and the last time an entity passed through the RECORD module. Thus, this option represents the time between arrivals to the module.

**Expression** The Expression option allows statistics to be collected on a general expression when the entity passes through the module.

In this situation, you can either use the Time Interval option as shown in Figure 7.8 or select the expression option and use  $\text{TNOW} - \text{myArriveToMakeRingTime}$  as the expression.

As shown in Figure 7.9, delete the appropriate connectors, place the ASSIGN module, fill out the ASSIGN module, and reconnect the modules. You should now delete the appropriate connectors and place a RECORD module as shown in Figure 7.8.



**Figure 7.8** Recording the make and inspect time.

In order to complete the example, you must ensure that the value of *vNumInMakeAndInspect* is incremented by 1 before the rings enter the Make Ring Process and decremented by 1 after the rings exit the Inspection process. The already placed ASSIGN modules can be used to accomplish this task. Within the Set Start Make Ring Time ASSIGN module, you should increment the variable as shown in Figure 7.10. Finally, in the Display Diameters ASSIGN module, you should decrement the variable as shown in Figure 7.11. This ensures that the variable *vNumInMakeAndInspect* correctly tracks how many rings are currently in the area of the process marked off between points F and G of Figure 7.6.

As mentioned at the beginning of the example, Arena™ will automatically collect statistical information about individual processes. Figure 7.12 shows the Statistical Collection area of the Run Setup dialog. You should make sure that the Process Check Box is checked. This will facilitate the comparison of what is collected automatically to what has been implemented in this example.

The final issue to be handled in this example is to specify the number of replications. First, a small pilot run will be performed to get initial half-widths, and then the required number of replications will be computed to ensure a 95% confidence interval with an error bound of  $\pm 20$  minutes. Using Run > Setup > Number of Replications, specify 10 as the number of replications and run the model.

The results for the combined process time based on the direct RECORD module are given in Figure 7.13. The results for the average number of pairs of rings in both processes are given in Figure 7.14. On an average, it takes about 365.79 minutes for all the rings to get through these two processes. This is occurring because all the rings are released at once to the Make Ring Process. If you examine Figure 7.15, you can see that the total time spent in each of the individual processes is automatically reported. Thus, based on this output, you can add up the process times from the process report. However, this does not automatically give a confidence interval on the total time. Based on the pilot run, the number of replications necessary to meet the confidence interval half-width target can be determined.

Using  $n \cong n_0(h_0/h)^2$  with  $n_0 = 10$ ,  $h_0 = 47.27$ , and  $h = 20$  indicates that approximately  $n = 56$  replications are needed to meet the criteria. If you wanted to use the iterative method,

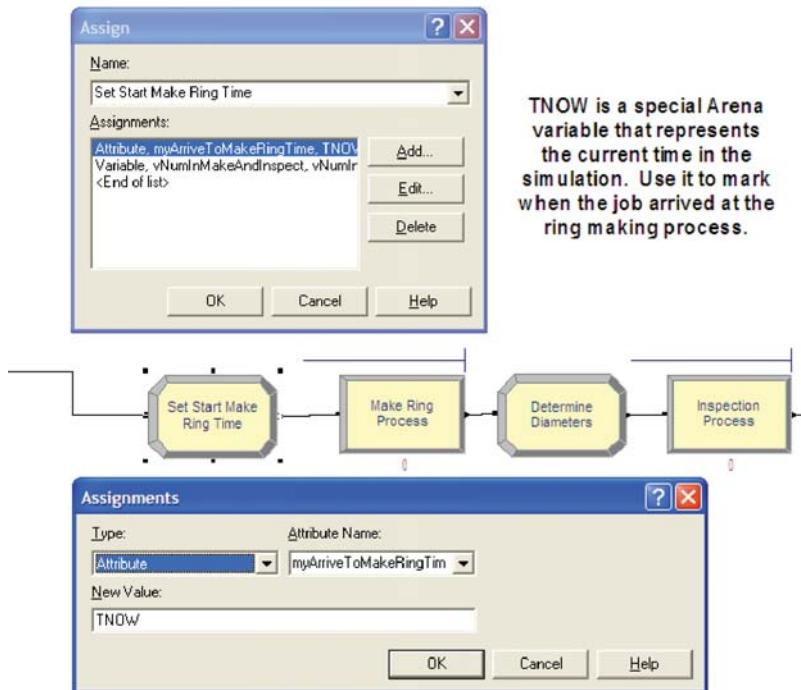


Figure 7.9 Capturing the start of ring making.

Assignments					
	Type	Variable Name	Attribute Name	New Value	
1	Attribute	Variable 1	myArriveToMakeRingTi	TNOW	
2	Variable	vNumInMakeAndInspect	Attribute 2	vNumInMakeAndInspect +1	

Figure 7.10 Set start Make Ring Time ASSIGN module.

Assignments				
	Type	Variable Name	New Value	
1	Variable	vNumInMakeAndInspect	vNumInMakeAndInspect - 1	
2	Variable	vID	myIDBigRing	
3	Variable	vOD	myODSmallRing	
4	Variable	vGap	myGap	

Figure 7.11 Display diameters ASSIGN module decrementing the number in make and inspect.

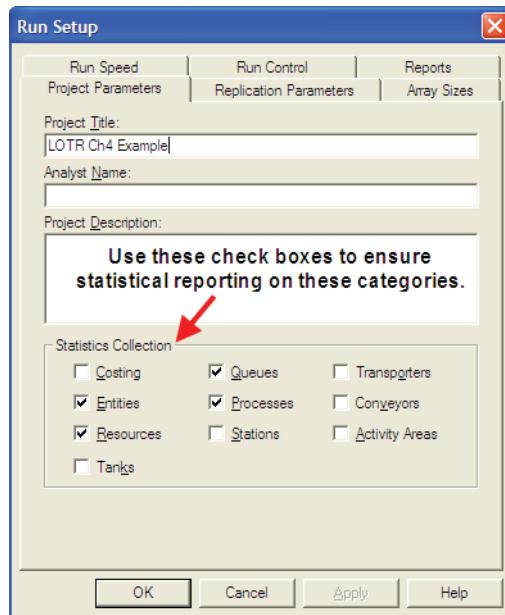


Figure 7.12 Run Setup statistical collection.



Figure 7.13 Result for RECORD module.



Figure 7.14 Results for average number in make and inspect.

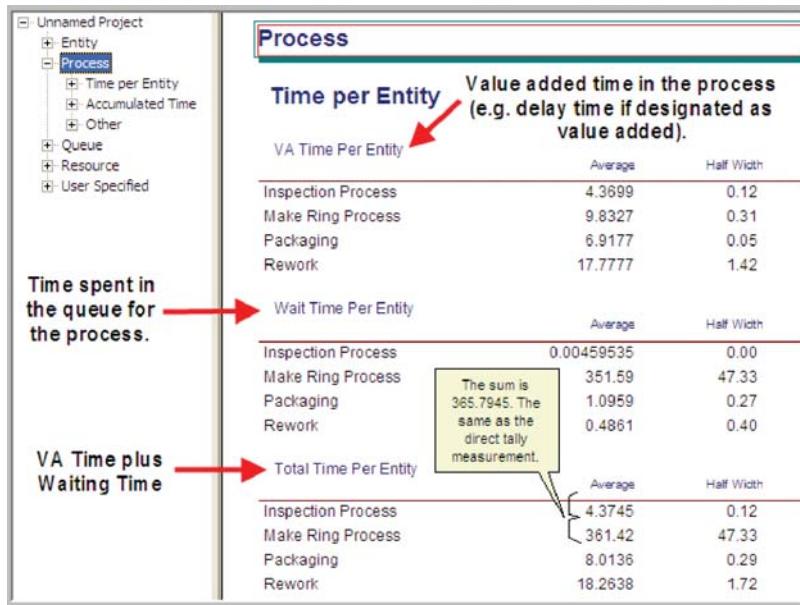


Figure 7.15 Arena's process statistical collection.

you must first determine the standard deviation from the pilot replications. In the case of multiple replications, you can use the half-width value and Equation (7.5) to compute  $s_0$ . For the example, this yields

$$s_0 = \frac{47.27\sqrt{10}}{t_{0.25,9}} = \frac{47.27\sqrt{10}}{2.262} = 66.083$$

Then, iteratively solving using the spreadsheet implementation for

$$h = t_{\alpha/2,n-1} \frac{s}{\sqrt{n}} \leq E$$

yields a recommended sample size of  $n = 44.4 \approx 45$ . The half-width ratio method tends to be more conservative in recommending the required sample size (number of replications).

The results in Figure 7.16 indicate that the half-width criterion is almost met by both sample sizes. Note that the make and inspection time is highly variable. The methods to determine the half-width assume that the standard deviation,  $s_0$ , in the pilot runs will be similar to the standard deviation observed in the final set of replications. However, when the full set of replications is run, the actual standard deviation may be different from that used in the pilot run. Thus, the half-width criterion might not be met. If the assumptions are reasonably met, there will be a high likelihood that the desired half-width will be very close to the desired criteria, as shown in this example.

### 7.3.3 Sequential Sampling for Finite-Horizon Simulations

The methods discussed for determining the sample size are based on predetermining a *fixed* sample size and then making the replications. If the half-width equation is considered as an

Interval	<b>n = 45</b>	Average	Half Width
Record Make and Inspect Time	400.83	20.50	
<hr/>			
Interval	<b>n = 56</b>	Average	Half Width
Record Make and Inspect Time	388.80	20.34	

Figure 7.16 Results from running  $n = 45$  and 56 replications.

iterative function of  $n$ :

$$h(n) = t_{\alpha/2,n-1} \frac{s(n)}{\sqrt{n}} \leq E \quad (7.10)$$

Then, it becomes apparent that additional replications of the simulation can be executed until the desired half-width bound is met. This is called sequential sampling, and in this case, the sample size of the experiment is not known in advance. The brute force method for implementing this approach would be to run and rerun the simulation each time increasing the number of replications until the criterion is met. By defining an OUTPUT statistic and using the ORUNHALF(*Output ID*) function, you can have the simulation stop when the criterion is met.

The pseudo-code for this concept is provided in Exhibit 7.1. First, create a logical entity at time 0.0. Then, use the special variable NREP to check if the current replication executing is less than or equal to 2, if it is dispose of the entity. If the current replication number is more than 2, then check if the half-width is less than or equal to the half-width bound. If it is, then indicate that the maximum number of replications, MREP, has been reached by setting MREP equal to the current replication number, NREP.

---

#### Exhibit 7.1 Pseudo-code for Sequential Sampling

---

```

1: CREATE a logical entity at time 0.0
2: IF NREP ≤ 2 THEN
3:   DISPOSE
4: ELSE
5:   IF half-width ≤ bound THEN
6:     ASSIGN MREP = NREP
7:   END IF
8: END IF
9: DISPOSE

```

---

In order to implement this pseudo-code, you must first define an OUTPUT statistic. This is done in Figure 7.17, where the function TAVG(*Tally ID*) has been used to get the end of replication average for the time spent in both the make and inspection processes. This ensures that you can use the ORUNHALF(*Output ID*) function. The ORUNHALF(*Output ID*) function returns the current half-width for the specified OUTPUT statistic based on all the replications that have been fully completed. This function can be used to check against the half-width bound.

Statistic - Advanced Process			
	Name	Type	Expression
1	TimeToMakeOrderStat	Output	TNOW
2	ProbOfOT	Output	TNOW > 960
3	AvgNumInMakeAndInspect	Time-Persistent	vNumInMakeAndInspect
4	MakeAndInspectTimeAvgAcrossRep	Output	TAVG(Record Make and Inspect Time)

Double-click here to add a new row.

Figure 7.17 OUTPUT statistic for record make and inspect time.

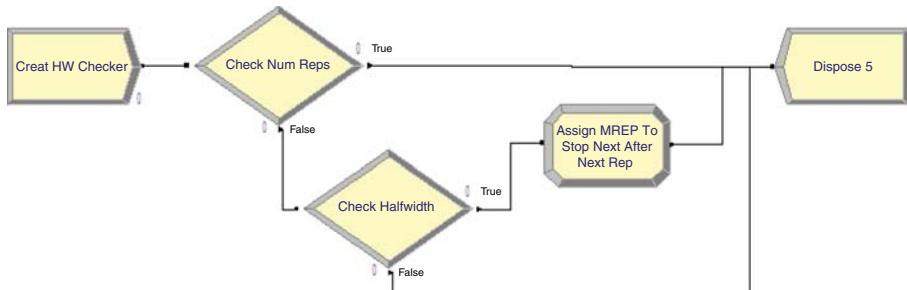


Figure 7.18 Logic for sequentially checking half-width.

Decide - Basic Process				
	Name	Type	If	Value
1	Check Num Reps	2-way by Condition	Expression	NREP <= 2
2	Check Halfwidth	2-way by Condition	Expression	ORUNHALF(MakeAndInspectTimeAvgAcrossRep) <= 20

Figure 7.19 DECIDE logic for sequential sampling.

Figure 7.18 illustrates the logic for implementing sequential sampling. The CREATE module creates a single entity at time 0.0 for each replication. That entity then proceeds to the Check Num Reps DECIDE module. The special variable, NREP, holds the *current* replication number. When NREP equals 1, no replications have yet been completed. When NREP equals 2, the first replication has been completed and the second replication is in progress. When NREP equals 3, two replications have been completed. At least two completed replications are needed to form a confidence interval (since the formula for the standard deviation has  $n - 1$  in its denominator). Since at least two replications are needed to start the checking, the entity can be disposed if the number of replications is less than or equal to 2. Once the simulation is past two replications, the entity will then begin checking the half-width.

The logic shown in Figure 7.19 shows that the second DECIDE module uses the function ORUNHALF(*Output ID*) to check the current half-width versus the error bound, which was 20 minutes in this case. If the half-width is less than or equal to the bound, the entity goes through the ASSIGN module to trigger the stopping of the replications. If the half-width is larger than the bound, the entity is disposed.

Assignments			
	Type	Other	New Value
1	Other	MREP	NREP
Double-click here to add a new row.			

Figure 7.20 Logic to stop the replication using MREP.

Interval	n = 51	Average	Half Width
Record Make and Inspect Time	395.51	19.62	

Figure 7.21 Output for sequential sampling.

The special variable called MREP represents the maximum number of replications to be run for the simulation. When you fill out the Run > Setup dialog and specify the number of replications, MREP is set to the value specified. At the beginning of each replication, NREP is checked against MREP. If NREP equals MREP, that replication will be the final replication run for the experiment. Therefore, if MREP is set equal to NREP in the check, the next replication will be the last replication. This actually causes one more replication to be executed than needed, but this cannot be avoided because of how the simulation executes. The only way to make this not happen is to use some VBA code. Figure 7.20 shows the ASSIGN module for setting MREP equal to NREP. Note that the assignment type is Other because MREP is a special variable and not a standard variable (as defined in the VARIABLE module).

This has been implemented in the file *LOTRCh7ExampleSequential.doe*. Before running the model, you should set the number of replications (MREP) in the Run > Setup > Replication Parameters dialog to some arbitrarily large integer. If you execute the model, you will get the result indicated in Figure 7.21.

In the sequential sampling experiment, there were 51 replications. This is actually less than the recommended 56 replications for the fixed half-width method in Figure 7.16, and the half-width met the criteria. The reason why the half-width in Figure 7.16 is larger than the half-width in Figure 7.21 is that replications 52–56 must have had significantly more variability. This is perfectly possible and emphasizes the fact that in the sequential sampling method, the number of replications is actually a random variable. If you were to use different streams and rerun the sequential sampling experiment, the number of replications completed may be different each time.

## 7.4 ANALYSIS OF INFINITE-HORIZON SIMULATIONS

This section discusses how to plan and analyze infinite-horizon simulations. When analyzing infinite-horizon simulations, the primary difficulty is the nature of within-replication data. In the finite-horizon case, the statistical analysis is based on three basic requirements:

1. Observations are independent
2. Observations are sampled from identical distributions
3. Observations are drawn from a normal distribution (or enough observations are present to invoke the central limit theorem)

These requirements were met by performing independent replications of the simulation to generate a random sample. In a direct sense, the outputs within a replication do not satisfy any of these requirements; however, certain procedures can be imposed on the manner in which the observations are gathered to ensure that these statistical assumptions are not grossly violated. The following will first explain why within-replication data typically violates these assumptions and then will provide some methods for mitigating the violations within the context of infinite-horizon simulations.

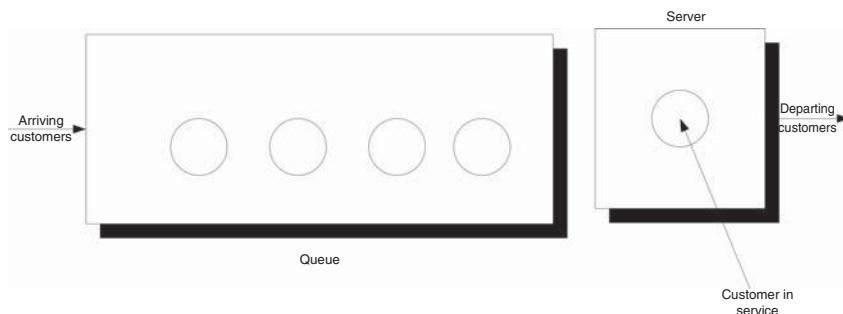
To illustrate the challenges related to infinite-horizon simulations, a simple spreadsheet simulation was developed for a M/M/1 queue as discussed in Chapter 4. The final spreadsheet is given in the spreadsheet file *MM1-QueueingSimulation.xls* that accompanies this chapter. The immediate feedback from a spreadsheet model should facilitate understanding the concepts. Consider a single-server queuing system as illustrated Figure 7.22.

For a single-server queuing system, there is an equation that allows the computation of the waiting times of each of the customers based on knowledge of the arrival and service times. Let  $X_1, X_2, \dots$  represent the successive service times and  $Y_1, Y_2, \dots$  represent the successive interarrival times for each of the customers that visit the queue. Let  $E[Y_i] = 1/\lambda$  be the mean of the interarrival times so that  $\lambda$  is the mean arrival rate. Let  $E[X_i] = 1/\mu$  be the mean of the service times so that  $\mu$  is the mean service rate. Let  $W_i$  be the waiting time in the queue for the  $i$ th customer. That is, the time between when the customer arrives until they enter service.

Lindley's equation, see Gross and Harris (1998), relates the waiting time to the arrivals and services as follows:

$$W_{i+1} = \max(0, W_i + X_i - Y_i) \quad (7.11)$$

The relationship says that the time that the  $(i + 1)$ th customer must wait is the time the  $i$ th waited, plus the  $i$ th customer's service time,  $X_i$  (because that customer is in front of the  $i$ th customer), is less than the time between arrivals of the  $i$ th and  $(i + 1)$ th customers,  $Y_i$ . If  $W_i + X_i - Y_i$  is less than zero, then the  $(i + 1)$ th customer arrived after the  $i$ th finished service, and thus the waiting time for the  $(i + 1)$ th customer is zero, because his service starts immediately.



**Figure 7.22** Single-server queuing system.

A	B	C	D	E	F	G	H
Customer Number	Waiting time	Service time	Interarrival time			Cumulative Sum	Cumulative Avg
n	W(n)	X(n)	Y(n)	W(n)+X(n)-Y(n)			
0	0	0.165049697	3.205865303	-3.040815606			
1	0	0.52814479	1.34872615	-0.82058136	0	0	
2	0	0.338501251	2.146635912	-1.808134661	0	0	
3	0	1.435505306	1.442529663	-0.007024357	0	0	
4	0	0.436372252	0.184805878	0.251566375	0	0	
5	0.251566375	0.600203813	0.576593493	0.275176695	0.251566375	0.050313275	
6	0.275176695	1.499580523	3.497237695	-1.722480477	0.52674307	0.087790512	
7	0	2.087126617	0.225136044	1.861990574	0.52674307	0.07524901	
8	1.861990574	0.940140227	4.372192785	-1.570061985	2.388733644	0.298591705	
9	0	0.55141614	0.710258739	-0.158842598	2.388733644	0.265414849	
10	0	0.489774047	0.0205816	0.469192446	2.388733644	0.238873364	

Figure 7.23 Spreadsheet for Lindley's equation.

A	B	C	D	E	F	G	H
Customer Number	Waiting time	Service time	Interarrival time		Cumulative Sum	Cumulative Avg	
n	W(n)	X(n)	Y(n)	W(n)+X(n)-Y(n)			
0	0	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C24+D24-E24			
1	=MAX(0,F24)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C25+D25-E25	=C25	=G25/B25	
2	=MAX(0,F25)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C26+D26-E26	=C26+G25	=G26/B26	
3	=MAX(0,F26)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C27+D27-E27	=C27+G26	=G27/B27	
4	=MAX(0,F27)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C28+D28-E28	=C28+G27	=G28/B28	
5	=MAX(0,F28)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C29+D29-E29	=C29+G28	=G29/B29	
6	=MAX(0,F29)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C30+D30-E30	=C30+G29	=G30/B30	
7	=MAX(0,F30)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C31+D31-E31	=C31+G30	=G31/B31	
8	=MAX(0,F31)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C32+D32-E32	=C32+G31	=G32/B32	
9	=MAX(0,F32)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C33+D33-E33	=C33+G32	=G33/B33	
10	=MAX(0,F33)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C34+D34-E34	=C34+G33	=G34/B34	

Figure 7.24 Spreadsheet formulas for queuing simulation.

Suppose that  $X_i \sim \exp(E[X_i] = 0.7)$  and  $Y_i \sim \exp(E[Y_i] = 1.0)$ . This is an M/M/1 queue with  $\lambda = 1$  and  $\mu = 10/7$ . Thus, using the equations given in Chapter 4 yields

$$\rho = 0.7 \quad (7.12)$$

$$L_q = \frac{0.7 \times 0.7}{1 - 0.7} = 1.6\bar{3} \quad (7.13)$$

$$W_q = \frac{L_q}{\lambda} = 1.6\bar{3} \text{ minutes} \quad (7.14)$$

The spreadsheet model involves generating  $X_i$  and  $Y_i$  as well as implementing Lindley's equation within the rows of the spreadsheet. Figure 7.23 shows some sample output from the simulation. The simulation was initialized with  $X_0 = 0$  and random draws for  $X_0$  and  $Y_0$ . The generated values for  $X_i$  and  $Y_i$  are based on the inverse transform technique for exponential random variables with the cell formulas given in cells D24 and E24, as shown in Figure 7.24.

	A	B
1		
2	Simple Queueing Simulation	
3		
4	Mean Service Time	0.7
5	Mean Interarrival time	1
6	Waiting time in Queue	1.633333333
7		
8	Sample Average	1.1872560
9	Sample Variance	3.4886921
10	StdDev	1.86780
11	Count	1000
12	StdError	0.0590652
13	conf level	0.95000
14	degrees of freedom	999.00000
15	alpha for mean CI	0.05000
16	t-value	1.962341416
17	half-width	0.11590599
18	CI Lower Limit for mean	1.07135003
19	CI Upper Limit for mean	1.30316201

**Figure 7.25** Results across 1000 customers.

Figure 7.24 shows that cell F25 is based on the value of cell F24. This implements the recursive nature of Lindley's formula. Column G holds the cumulative sum:

$$\sum_{i=1}^n W_i \text{ for } n = 1, 2, \dots$$

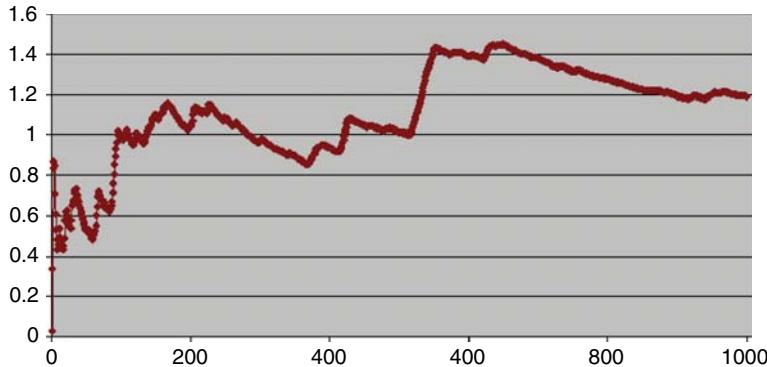
and column H holds the cumulative average

$$\frac{1}{n} \sum_{i=1}^n W_i \text{ for } n = 1, 2, \dots$$

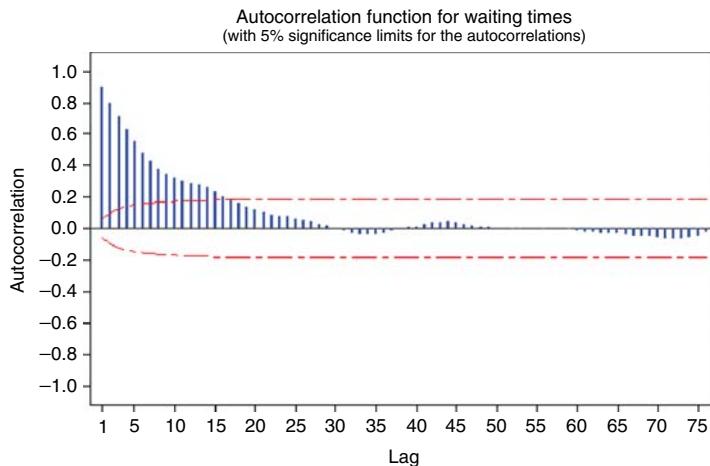
Thus, cell H26 is the average of the first two customers, cell H27 is the average of the first 3 customers, and so forth. The final cell of column H represents the average of all the customers' waiting times.

Figure 7.25 shows the results of the simulation across 1000 customers. The analytical results indicate that the true long-run expected waiting time in the queue is 1.633 minutes. The average over the 1000 customers in the simulation is 1.187 minutes. Figure 7.25 indicates that the sample average is significantly lower than the true expected average. Figure 7.26 presents the cumulative average plot of the first 1000 customers. As seen in the plot, the cumulative average starts out low and then eventually trends toward 1.2 minutes.

The first issue to consider with this data is independence. To do this, you should analyze the 1000 observations in terms of its autocorrelation. From Figure 7.27, it is readily apparent that the data has strong positive correlation using the methods discussed in Chapter 3. The lag-1 correlation for this data is estimated to be about 0.9. Figure 7.27 clearly indicates the strong first-order linear dependence between  $W_i$  and  $W_{i-1}$ . This positive dependence implies that if the previous customer waited a long time, the next customer is likely to wait



**Figure 7.26** Cumulative average waiting time of 1000 customers



**Figure 7.27** Autocorrelation plot for waiting times.

a long time. If the previous customer had a short waiting time, then the next customer is likely to have a short waiting time. This makes sense with respect to how a queue operates.

Strong positive correlation has serious implications when developing confidence intervals on the mean customer waiting time because the usual estimator for the sample variance

$$S^2(n) = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2 \quad (7.15)$$

is a biased estimator for the true population variance when there is correlation in the observations. This issue will be reexamined when ways to mitigate these problems are discussed.

The second issue that needs to be discussed is that of the non-stationary behavior of the data. As discussed in Chapter 6, non-stationary data indicates some dependence on time. More generally, non-stationary implies that the  $W_1, W_2, W_3, \dots, W_n$  are not obtained from identical distributions.

Why should the distribution of  $W_1$  not be the same as the distribution of  $W_{1000}$ ? The first customer is likely to enter the queue with no previous customers present and thus it is very likely that the first customer will experience little or no wait (the way  $W_0$  initialized in this example allows a chance of waiting for the first customer). However, the 1000th customer may face an entirely different situation. Between the 1st and the 1000th customer, there might likely be a line formed. In fact from the M/M/1 formula, it is known that the steady-state expected number in the queue is 1.633. Clearly, the conditions that the 1st customer faces are different from the 1000th customer. Thus, the distributions of their waiting times are likely to be different.

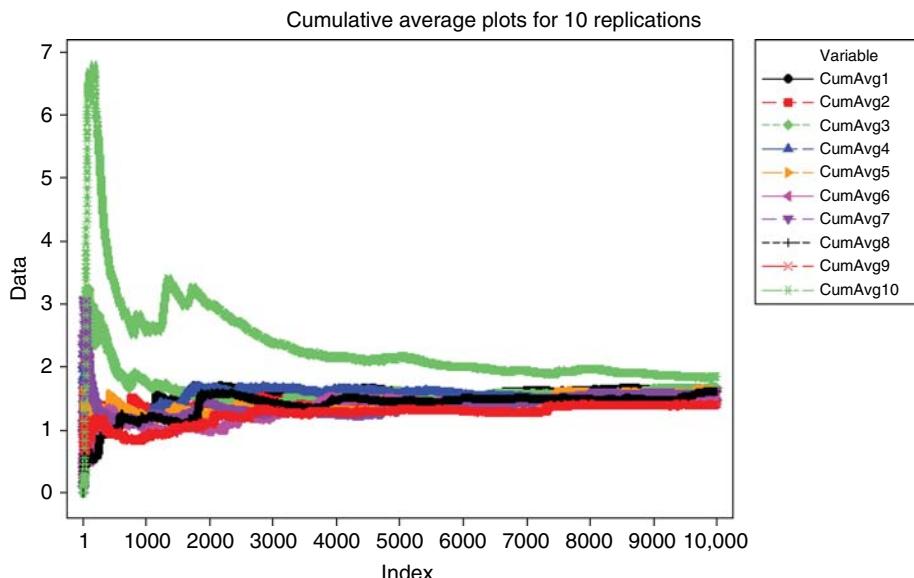
Recall the definition of covariance stationary from Chapter 2. A time series,  $W_1, W_2, W_3, \dots, W_n$  is said to be *covariance stationary* if:

- The mean exists and  $\theta = E[X_i]$ , for  $i = 1, 2, \dots, n$
- The variance exists and  $\text{Var}[X_i] = \sigma^2 > 0$ , for  $i = 1, 2, \dots, n$
- The lag- $k$  autocorrelation,  $\rho_k = \text{cor}(X_i, X_{i+k})$ , is not a function of  $i$ , that is, the correlation between any two points in the series does not depend upon where the points are in the series, it depends only upon the distance between them in the series.

In the case of the customer waiting times, you should conclude from the discussion that it is very likely that  $\theta \neq E[X_i]$  and  $\text{Var}[X_i] \neq \sigma^2$  for each  $i = 1, 2, \dots, n$  for the time series.

Do you think that is it likely that the distributions of  $W_{9999}$  and  $W_{10,000}$  will be similar? The argument that the 9999th customer is on an average likely to experience similar conditions as the 10,000th customer sure seems reasonable. Figure 7.28 shows 10 different replications of the cumulative average for a 10,000-customer simulation. This was developed using the sheet labeled 10,000-customers.

From the figure, you can see that the cumulative average plots can vary significantly over the 10,000 customers with the average tracking above the true expected value, below



**Figure 7.28** Multiple sample paths of queuing simulation.

the true expected value, and possibly toward the true expected value. Each of these plots was generated by pressing the F9 function key to have the spreadsheet recalculate with new random numbers and then capturing the observations onto another sheet for plotting. Essentially, this is like running a new replication. You are encouraged to try generating multiple sample paths using the provided spreadsheet. For the case of 10,000 customers, you should notice that the cumulative average starts to approach the expected value of the steady-state mean waiting time in the queue with increasing number of customers. This is the law of large numbers in action. It appears that it takes a period of time for the performance measure to *warm up* toward the true mean. Determining the warm-up time will be the basic way to mitigate the problem of nonidentical distributions.

From this discussion, you should conclude that the second basic statistical assumption of identically distributed data is not valid for within-replication data. From this, you can also conclude that it is very likely that the data are not normally distributed. In fact, for the M/M/1, it can be shown that the steady-state distribution for the waiting time in the queue is not a normal distribution. Thus, all three of the basic statistical assumptions are violated for the within-replication data of this example. This problem needs to be addressed in order to properly analyze infinite-horizon simulations.

There are two basic methods for performing infinite-horizon simulations. The first is to perform multiple replications. This approach addresses independence and normality in a similar manner as the finite-horizon case, but special procedures will be needed to address the non-stationary aspects of the data. The second basic approach is to work with one very long replication. Both of these methods depend on first addressing the problem of the non-stationary aspects of the data. The next section looks at ways to mitigate the non-stationary aspect of within-replication data for infinite-horizon simulations.

#### 7.4.1 Assessing the Effect of Initial Conditions

Consider the output stochastic process  $X_i$  of the simulation. Let  $F_i(x|I)$  be the conditional cumulative distribution function of  $X_i$  where  $I$  represents the initial conditions used to start the simulation at time 0. If  $F_i(x|I) \rightarrow F(x)$  when  $i \rightarrow \infty$ , for all initial conditions  $I$ , then  $F(x)$  is called the steady-state distribution of the output process [Law, 2007].

In infinite-horizon simulations, estimating parameters of the steady-state distribution,  $F(x)$ , such as the steady-state mean,  $\theta$ , is often the key objective. The fundamental difficulty associated with estimating steady-state performance is that unless the system is initialized using the steady-state distribution (which is not known), there is no way to directly observe the steady-state distribution.

It is true that if the steady-state distribution exists and you run the simulation long enough the estimators will tend to converge to the desired quantities. Thus, within the infinite-horizon simulation context, you must decide on how long to run the simulations and how to handle the effect of the *initial conditions* on the estimates of performance. The initial conditions of a simulation represent the state of the system when the simulation is started. For example, in simulating the pharmacy system, the simulation was started with no customers in service or in the line. This is referred to as *empty and idle*. The initial conditions of the simulation affect the rate of convergence of estimators of steady-state performance.

Because the distributions  $F_i(x|I)$  at the start of the replication tend to depend more heavily upon the initial conditions, estimators of steady-state performance such as the sample average,  $\bar{X}$ , will tend to be *biased*. A point estimator,  $\hat{\theta}$ , is an *unbiased* estimator of the

**TABLE 7.1 Ten Replications of 20 Customers**

r	$\bar{W}_r$	$B_r = \bar{W}_r - W_q$
1	0.194114	-1.43922
2	0.514809	-1.11852
3	1.127332	-0.506
4	0.390004	-1.24333
5	1.05056	-0.58277
6	1.604883	-0.02845
7	0.445822	-1.18751
8	0.610001	-1.02333
9	0.52462	-1.10871
10	0.335311	-1.29802
$\bar{\bar{W}} = 0.6797$		$\bar{B} = -0.9536$

parameter of interest,  $\theta$ , if  $E[\hat{\theta}] = \theta$ . That is, if the expected value of the sampling distribution is equal to the parameter of interest, then the estimator is said to be unbiased. If the estimator is biased, then the difference,  $E[\hat{\theta}] - \theta$ , is called the bias of the estimator  $\hat{\theta}$ .

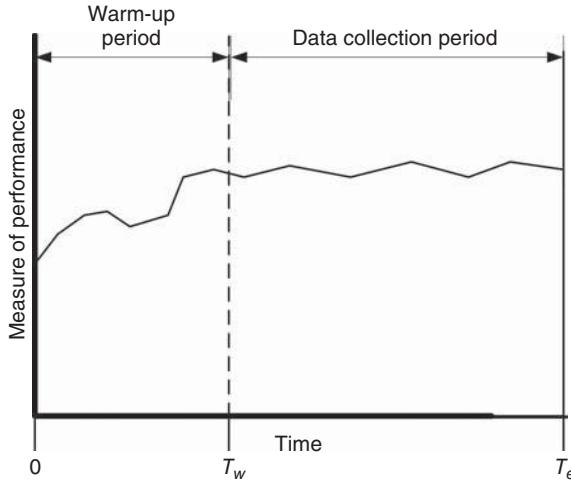
Note that any individual difference between the true parameter,  $\theta$ , and a particular observation,  $X_i$ , is called error,  $e_i = X_i - \theta$ . If the expected value of the errors is not zero, then there is bias. A particular observation is not biased. Bias is a property of the estimator. Bias is analogous to being consistently off target when shooting at a bulls-eye. It is as if the sights on your gun are crooked. In order to estimate the bias of an estimator, you must have multiple observations of the estimator. Suppose that you are estimating the mean waiting time in the queue as per the previous example and that the estimator is based on the first 20 customers. That is, the estimator is

$$\bar{W}_r = \frac{1}{20} \sum_{i=1}^{20} W_{ir}$$

and there are  $r = 1, 2, \dots, 10$  replications. Table 7.1 shows the sample average waiting time for the first 20 customers for 10 different replications. In the table,  $B_r$  is an estimate of the bias for the  $r$ th replication, where  $W_q = 1.633$ . Upon averaging across the replications, it can be seen that  $\bar{B} = -0.9536$ , which indicates that the estimator based only on the first 20 customers has significant negative bias, that is, on an average it is less than the target value.

This is the so-called *initialization bias problem* in steady-state simulation. Unless the initial conditions of the simulation can be generated according to  $F(x)$ , which is not known, you must focus on methods that detect and/or mitigate the presence of initialization bias. Tests for initialization bias are described in Schruben [1982] and Schruben et al. [1983].

One strategy for initialization bias mitigation is to find an index,  $d$ , for the output process,  $X_i$ , so that  $X_i; i = d + 1, \dots$  will have substantially similar distributional properties as the steady-state distribution  $F(x)$ . This is called the simulation warm-up problem, where  $d$  is called the warm-up point and  $i = 1, \dots, d$  is called the warm-up period for the simulation. Then the estimators of steady-state performance are based only on  $X_i; i = d + 1, \dots$



**Figure 7.29** The concept of the warm-up period.

For example, when estimating the steady-state mean waiting time for each replication  $r$ , the estimator would be

$$\bar{W}_r = \frac{1}{n-d} \sum_{i=d+1}^n W_{ir} \quad (7.16)$$

For time-based performance measures, such as the average number in queue, a time  $T_w$  can be determined past which the data collection process can begin. Estimators of time-persistent performance such as the sample average are computed as

$$\bar{Y}_r = \frac{1}{T_e - T_w} \int_{T_w}^{T_e} Y_r(t) dt \quad (7.17)$$

Figure 7.29 shows the concept of a warm-up period for a simulation replication. When you perform a simulation, you can easily specify a time-based warm-up period using the Run > Setup > Replication Parameters panel. In fact, even for observation-based data, it will be more convenient to specify the warm-up period in terms of time. A given value of  $T_w$  implies a particular value of  $d$  and vice a versa. Specifying a warm-up period causes an event to be scheduled for time  $T_w$ . At that time, all the accumulated statistical counters are cleared so that the net effect is that statistics are only collected over the period from  $T_w$  to  $T_e$ . The problem then becomes that of finding an appropriate warm-up period.

Before proceeding with how to assess the length of the warm-up period, the concept of steady state needs to be further examined. This subtle concept is often misunderstood or misrepresented. Often you will hear the phrase: *The system has reached steady state*. The correct interpretation of this phrase is that the distribution of the desired performance measure has reached a point where it is sufficiently similar to the desired steady-state distribution. Steady state is a concept involving the performance measures generated by the system as time goes to infinity. However, sometimes this phrase is interpreted incorrectly to mean that the system *itself* has reached steady state. Let me state emphatically that the system *never* reaches steady state. If the system itself reached steady state, then by implication it would never change with respect to time. It should be clear that the system continues to

evolve with respect to time; otherwise, it would be a very boring system! Thus, it is incorrect to indicate that the system has reached steady state. Because of this, do not use the phrase: *The system has reached steady state.*

Understanding this subtle issue raises an interesting implication concerning the notion of deleting data to remove the initialization bias. Suppose that the state of the system at the end of the warm-up period,  $T_w$ , is exactly the same as that at  $T = 0$ . For example, it is certainly possible that at time  $T_w$  for a particular replication that the system was empty and idle. Since the state of the system at  $T_w$  is the same as that of the initial conditions, there will be no effect of deleting the warm-up period for this replication. In fact there will be a negative effect, in the sense that data will have been thrown away for no reason. Deletion methods are predicated on the likelihood that the state of the system seen at  $T_w$  is more representative of steady-state conditions. At the end of the warm-up period, the system can be in *any of the possible* states of the system. Some states will be more likely than others. If multiple replications are made, then at  $T_w$ , each replication will experience a different set of conditions at  $T_w$ . Let  $I_{T_w}^r$  be the initial conditions (state) at time  $T_w$  on replication  $r$ . By setting a warm-up period and performing multiple replications, you are in essence sampling from the distribution governing the state of the system at time  $T_w$ . If  $T_w$  is long enough, then on average across the replications, you are more likely to start collecting data when the system is in states that are more representative over the long term (rather than just empty and idle).

Many methods and rules have been proposed to determine the warm-up period. The interested reader is referred to Wilson and Pritsker [1978], Lada et al. [2003], Litton and Harmonosky [2002], White et al. [2000], Cash et al. [1992], and Rossetti and Delaney [1995] for an overview of such methods. This discussion will concentrate on the visual method proposed in Welch [1983a] and Welch [1983b].

The basic idea behind Welch's graphical procedure is simple:

- Make  $R$  replications. Typically,  $R \geq 5$  is recommended.
- Let  $Y_{rj}$  be the  $j$ th observation on replication  $r$  for  $j = 1, 2, \dots, m_r$ , where  $m_r$  is the number of observations in the  $r$ th replication and  $r = 1, 2, \dots, n$ .
- Compute the averages across the replications for each  $j = 1, 2, \dots, m$ , where  $m = \min(m_r)$  for  $r = 1, 2, \dots, n$ .

$$\bar{Y}_{.j} = \frac{1}{n} \sum_{r=1}^n Y_{rj}$$

- Plot  $\bar{Y}_{.j}$  for each  $j = 1, 2, \dots, m$ .
- Apply smoothing techniques to  $\bar{Y}_{.j}$  for  $j = 1, 2, \dots, m$ .
- Visually assess where the plots start to converge.

Let us apply Welch's procedure to the replications generated from the Lindley equation simulation. Using the 10 replications stored on sheet *10Replications*, compute the average across each replication for each customer. In Figure 7.30, cell B2 represents the average across the 10 replications for the first customer. Column D represents the cumulative average associated with column B.

Figure 7.31 is the plot of the cumulative average (column D) superimposed on the averages across replications (column B). The cumulative average is one method of smoothing

B2	$=AVERAGE(E2:N2)$								
	A	B	C	D	E	F	G	H	I
1		Average	Sum	CumAvg	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5
2		0.548433	0.548433	0.548432719	0	0.337306	0	1.79165	0.716249
3		0.359163	0.907596	0.453798017	0.213726	0.427596	0	1.892545	0
4		0.793544	1.70114	0.567046583	0.428974	1.498441	0	2.921952	1.373886
5		0.679799	2.380939	0.595234777	0.141178	2.265391	0	1.559509	1.523948
6		0.772271	3.15321	0.630642059	0.255458	0.839788	0	1.930919	1.569935
7		0.598509	3.75172	0.625286585	0.022953	0.909185	0	1.677104	1.460226
8		0.760241	4.51196	0.644565755	0	0.917162	0	0.152849	1.979963
9		0.879366	5.391326	0.673915781	0	0.360169	0	0	3.596641
10		1.169247	6.560573	0.728952544	0.161749	0.457508	0	0	2.767113
11		0.897993	7.458566	0.745856589	0.351444	1.45561	0	0	1.213456

Figure 7.30 Computing the averages for the Welch plot.

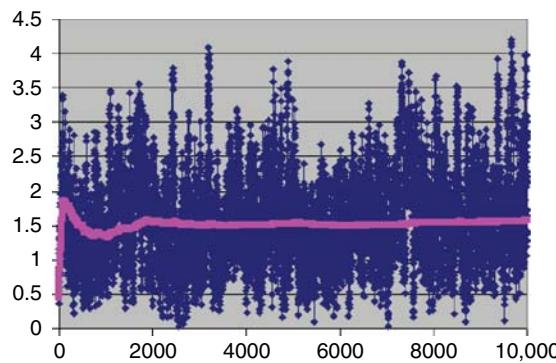


Figure 7.31 The Welch plot with superimposed cumulative average line.

the data. From the plot, you can infer that after about 3000 customers, the cumulative average has started to converge. Thus, from this analysis, you might infer that  $d = 3000$ .

When you perform an infinite-horizon simulation by specifying a warm-up period and making multiple replications, you are using the method of *replication–deletion*. If the method of replication–deletion with  $d = 3000$  is used for the current example, a slight reduction in the bias can be achieved as indicated in Table 7.2.

While not definitive for this simple example, the results suggest that deleting the warm-up period helps to reduce initialization bias. This model’s warm-up period will be further analyzed using additional tools available in the next section.

In performing the method of replication–deletion, there is a fundamental trade-off that occurs. Because data is deleted, the variability of the estimator will tend to increase, while the bias will tend to decrease. This is a trade-off between a reduction in bias and an increase in variance. That is, accuracy is being traded off against precision when deleting the warm-up period. In addition to this trade-off, data from each replication is also being thrown away. This takes computational time that could be expended more effectively on collecting usable data. Another disadvantage of performing replication–deletion is that the techniques for assessing the warm-up period (especially graphical) may require significant data storage. The Welch plotting procedure requires the saving of data points for post-processing after the simulation run. In addition, significant time by the analyst may be required to perform the technique and the technique is subjective.

**TABLE 7.2 Replication–Deletion Results,  $d = 3000$** 

$r$	$\overline{W}_r(d = 0)$	$\overline{W}_r(d = 3000)$	$B_r(d = 0)$	$B_r(d = 3000)$
1	1.594843	1.592421	-0.03849	-0.04091
2	1.452237	1.447396	-0.1811	-0.18594
3	1.657355	1.768249	0.024022	0.134915
4	1.503747	1.443251	-0.12959	-0.19008
5	1.606765	1.731306	-0.02657	0.097973
6	1.464981	1.559769	-0.16835	-0.07356
7	1.621275	1.75917	-0.01206	0.125837
8	1.600563	1.67868	-0.03277	0.045347
9	1.400995	1.450852	-0.23234	-0.18248
10	1.833414	1.604855	0.20008	-0.02848
	$\overline{\overline{W}} = 1.573617$	$\overline{\overline{W}} = 1.603595$	$\overline{B} = -0.05972$	$\overline{B} = -0.02974$
	$s = 0.1248$	$s = 0.1286$	$s = 0.1248$	$s = 0.1286$
95% LL	1.4843	1.5116	-0.149023	-0.121704
95% UL	1.6629	1.6959	-0.029590	0.062228

When a simulation has many performance measures, you may have to perform a warm-up period analysis for every performance measure. This is particularly important, since, in general, the performance measures of the same model may converge toward steady-state conditions at different rates. In this case, the length of the warm-up period must be sufficiently long enough to cover all the performance measures. Finally, replication–deletion may simply compound the bias problem if the warm-up period is insufficient relative to the length of the simulation. If you have not specified a long enough warm-up period, you are potentially compounding the problem for  $n$  replications.

Despite all these disadvantages, replication–deletion is very much used in practice because of the simplicity of the analysis after the warm-up period has been determined. Once you are satisfied that you have a good warm-up period, the analysis of the results is the same as that of finite-horizon simulations. Replication–deletion also facilitates the use of experimental design techniques that rely on replicating design points.

In addition, replication–deletion facilitates the use of such tools as the Process Analyzer and OptQuest. The Process Analyzer will be discussed later in this chapter and OptQuest in subsequent chapters. The next section illustrates how to perform the method of replication–deletion on this simple M/M/1 model.

#### 7.4.2 Performing the Method of Replication–Deletion

The first step in performing the method of replication–deletion is to determine the length of the warm-up period. This example illustrates how to

- save the values from observation and time-based data to files within the model for post-processing within the Output Analyzer,
- assess the warm-up period using the Output Analyzer,
- make the Welch plots based on the saved data using Excel and Microsoft Access,
- set up and run the multiple replications,
- interpret the results.

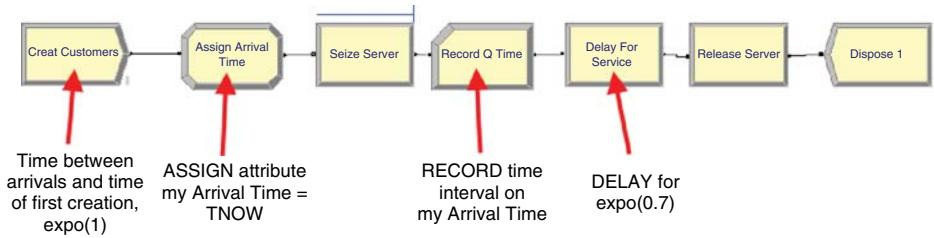


Figure 7.32 M/M/1 model.

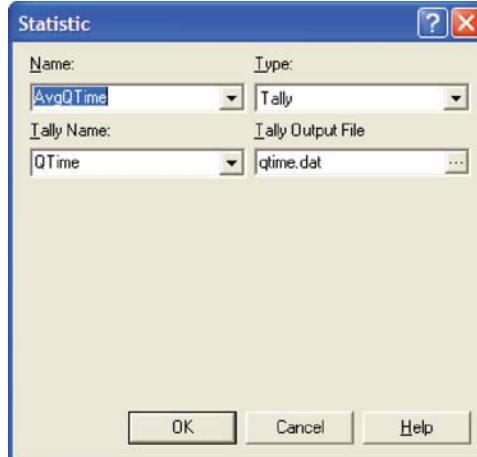
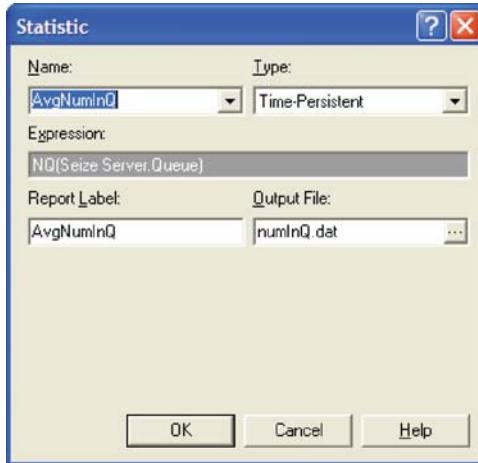


Figure 7.33 Capturing tally-based data to a file.

The file *CH7-MM1-ReplicationDeletion.doe* contains the M/M/1 model for which the time between arrivals  $\exp(E[Y_i] = 1.0)$  and the service times  $\exp(E[Y_i] = 0.7)$  are set in the CREATE and DELAY modules, respectively. Figure 7.32 shows the overall flow of the model. Since the waiting times in the queue need to be captured, an ASSIGN module has been used to mark the time that the customer arrives to the queue. Once the customer exits the SEIZE module, they have been removed from the queue to start service. At that point, a RECORD module is used to capture the time interval representing the queuing time.

Using the STATISTIC data module, the statistics collected within the replications can be captured to files for post-processing by the Output Analyzer. This example will collect the queue time of each customer and the number of customers in the queue over time. Figure 7.33 shows how to add a statistic using the STATISTIC data module that will capture observation-based data (e.g., queue time) to a file. The Type of the statistic is Tally and the Tally Name corresponds to the name specified in the RECORD module that is used to capture the observations. When a name for the output file is specified, the model will store every observation and the time of the observation to a file with the extension (.dat). These files are not human readable but can be processed by the Output Analyzer.

Figure 7.34 shows how to create a statistic to capture the number in queue over time to a file. The type of the statistic is indicated as time persistent and an expression that represents the value of the quantity to be observed through time must be given. In this case, the expression builder has been used to select the current number of entities in the Seize



**Figure 7.34** Capturing time-persistent data to a file.

Server.Queue (using the NQ(queue ID) function). When the Output File is specified, the value of the expression and the time of the observation are written to a (.dat) file.

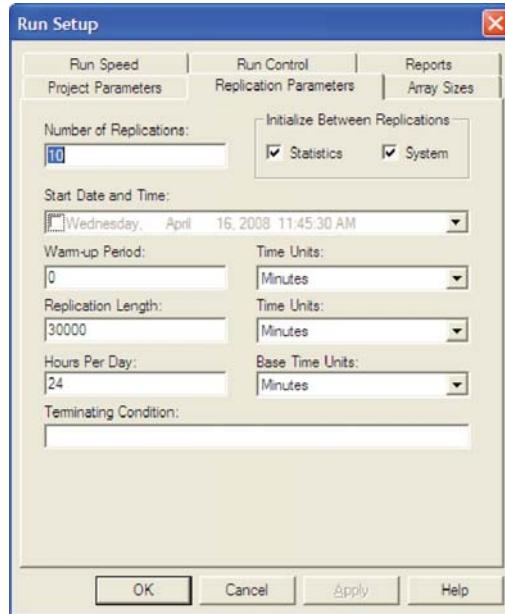
When performing a warm-up period analysis, the first decision to make is the length of each replication. In general, there is very little guidance that can be offered other than to try different run lengths and check for the sensitivity of your results. Within the context of queuing simulations, the work by Whitt [1989] offers some ideas on specifying the run length, but these results are difficult to translate to general simulations.

Since the purpose here is to determine the length of the warm-up period, then the run length should be bigger than what you suspect the warm-up period to be. In this analysis, it is better to be conservative. You should make the run length as long as possible, given your time and data storage constraints. Banks et al. [2005] offered the rule of thumb that the run length should be at least 10 times the amount of data deleted, that is,  $n \geq 10d$  or in terms of time  $T_e \geq 10T_w$ . Of course, this is a “catch 22” situation because you need to specify  $n$  or equivalently  $T_e$  in order to assess  $T_w$ . Setting  $T_e$  very large is recommended when doing a preliminary assessment of  $T_w$ . Then, you can use the rule of thumb of 10 times the amount of data deleted when doing a more serious assessment of  $T_w$  (e.g., using the Welch plots).

A preliminary assessment of the current model has already been performed based on the previously described Excel simulation. That assessment suggested a deletion point of at least  $d = 3000$  customers. This can be used as a starting point in the current effort. Now,  $T_w$  needs to be determined based on  $d$ . The value of  $d$  represents the customer number for the end of the warm-up period. To get  $T_w$ , you need to answer the question: How long (on an average) will it take for the simulation to generate observations. In this model, the mean number of arrivals is one customer per minute. Thus, the initial  $T_w$  is

$$3000 \text{ customers} \times \frac{\text{Minute}}{1 \text{ customer}} = 3000 \text{ minutes} \quad (7.18)$$

and, therefore, the initial  $T_e$  should be 30,000 minutes. Specify 30,000 minutes for the replication length and 10 replications on the Run > Setup > Replication Parameters as shown in Figure 7.35.



**Figure 7.35** Replication Parameters tab.

The replication length is the time when the simulation will end. The Base Time Units can be specified for how the report statistics on the default reports and represents the units for TNOW. The Time Units for the Warm-up Period and the Replication Length can also be set. The Warm-up period is the time where the statistics will be cleared. Thus, data will be reported over a net (Replication Length - Warm-up Period) time units. Running the simulation will generate two files *numInQ.dat* and *qtime.dat* within the current working directory for the model.

#### 7.4.3 Looking for the Warm-Up Period in the Output Analyzer

In order to analyze the situation using the Output Analyzer open up the Output Analyzer and create a new data group (Figure 7.36). Then, you should add the two files *numInQ.dat* and *qtime.dat* to the data group.

As previously discussed, the Welch plot is a reasonable approach to assessing the warm-up period. Unfortunately, the Output Analyzer does not automatically perform the Welch plot analysis. The best that can be done with the Output Analyzer is to look at each replication individually using the Moving Average command from the Plot menu as shown in Figure 7.37.

In this command, the Output Analyzer allows data to be smoothed for the replication selected by the moving average, exponential, or cumulative options. The Moving Average option allows a moving average smoothing technique to be applied to the data. The exponential option allows an exponential moving average technique to be applied. The cumulative option superimposes the cumulative average on the plot of the data. Figure 7.38 shows the cumulative average plot for the first replication for the waiting times in the queue. From this plot, it appears that the warm-up period is at least 4000 time units (minutes); however, this is just one replication. You can make a plot for each replication and try to eye-ball the

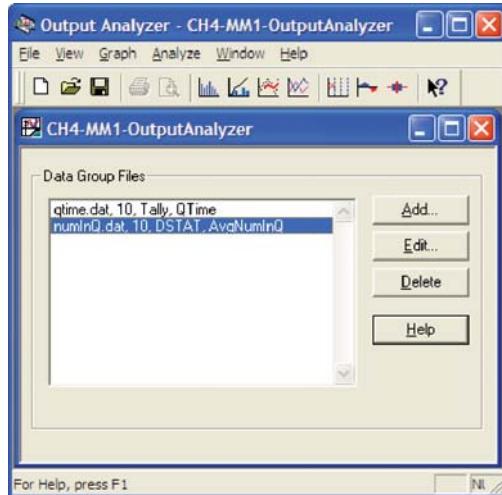


Figure 7.36 Output Analyzer data group.

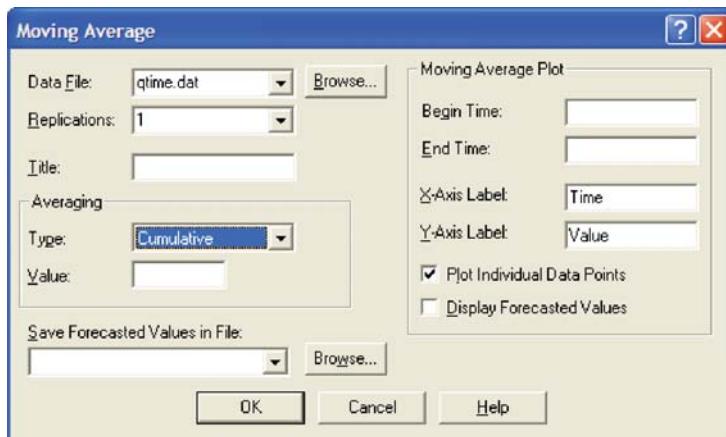
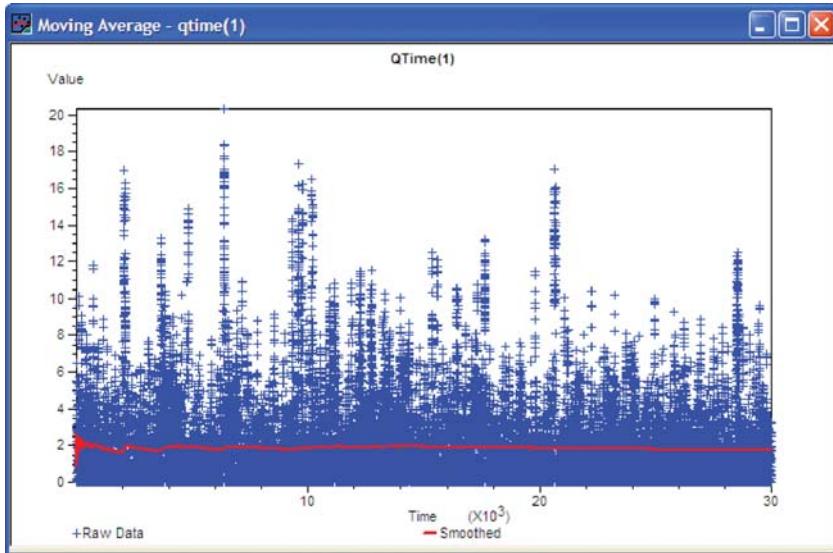


Figure 7.37 Moving average option with cumulative average selected.

warm-up time on each plot. This approach can also be applied to time-persistent data after *filtering* the data into batches.

Time-persistent observations are saved within a file from within the model such that the time of the observation and the value of the state variable at the time of change are recorded. Thus, the observations are not equally spaced in time. In order to apply, the moving average command you need to cut the data into discrete equally spaced intervals of time as illustrated in Figure 7.7. Suppose that you divide  $T_e$  into  $k$  intervals of size  $\Delta t$ , so that  $T_e = k \times \Delta t$ . The time average over the  $j$ th interval is given by

$$\bar{Y}_{rj} = \frac{1}{\Delta t} \int_{(j-1)\Delta t}^{j\Delta t} Y_r(t) dt$$



**Figure 7.38** Cumulative average plot of first replication.

Thus, the overall time average can be computed from the time average associated with each interval as shown below:

$$\begin{aligned}\bar{Y}_r &= \frac{\int_0^{T_e} Y_r(t) dt}{T_e} = \frac{\int_0^{T_e} Y_r(t) dt}{k\Delta t} \\ &= \frac{\sum_{j=1}^k \int_{(j-1)\Delta t}^{j\Delta t} Y_r(t) dt}{k\Delta t} = \frac{\sum_{j=1}^k \bar{Y}_{rj}}{k}\end{aligned}$$

Each of the  $\bar{Y}_{rj}$  are computed over intervals of time that are equally spaced and can be treated as if they are tally-based data.

The computation of the  $\bar{Y}_{rj}$  for time-persistent data can be achieved within the Output Analyzer by using the Batch/Truncate Obs's option in the Analyze menu (Figure 7.39). Figure 7.40 illustrates the Batch/Truncate dialog. Using this dialog, you can select which replication that you want to batch, the type of batching (time based or observation based), and the size of the batch (either in time units or in number of observations).

Since the number in queue data is time persistent, time-based batches are selected, and the batch size is specified in terms of time. In this case, the data is being batch based on a time interval of 10 minutes.

This produces a file *numInQfilter.flt* which contains the  $\bar{Y}_{rj}$  as observations. This file can then be added to the Output Analyzer's data group and processed using the Moving Average command from the Plot menu as previously illustrated. The resulting plot is show in Figure 7.41.

You can make a plot for each replication and try to eye-ball the warm-up time on each plot; however, as previously discussed, the Welch plot may be more effective in helping this process. To make the Welch plot, you need to process the data outside of the Output Analyzer in order to compute the averages across the replications.

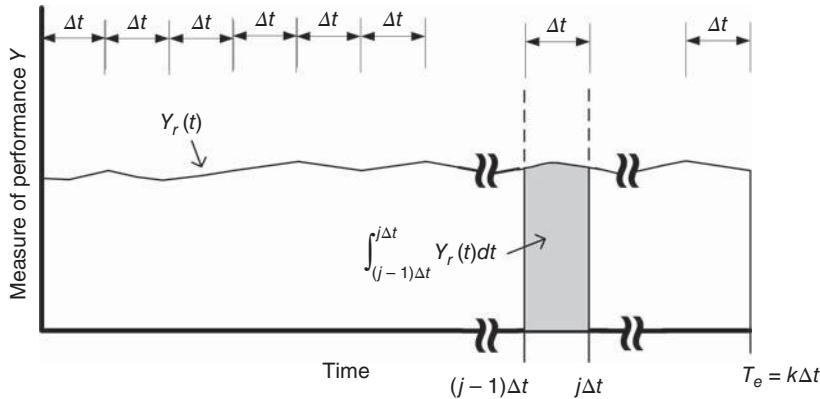


Figure 7.39 Time-persistent data.

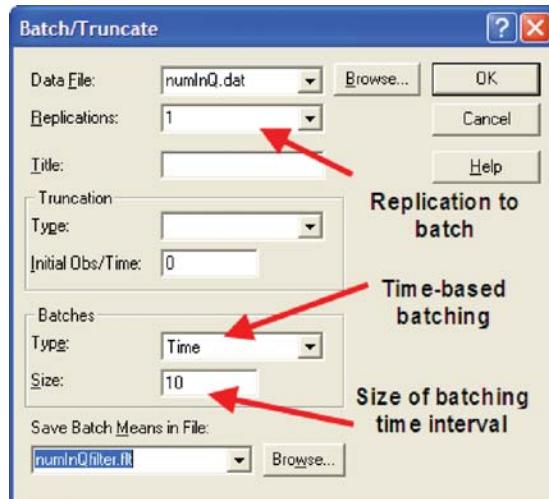


Figure 7.40 Batch/Truncate dialog.

In order to process the recorded data from the simulation outside of Arena<sup>TM</sup>, you must direct the program to produce a text file rather than a *dat* file. There are two changes that need to be made to facilitate this process. The first is to that output files should be written as text using the Run Setup > Run Control > Advanced dialog as shown in Figure 7.42. The second is to change the file extension to “csv” within the STATISTICS module as shown in Figure 7.43. This will facilitate Excel recognizing the file as a comma separate variable (CSV) file. Thus, when you double-click on the file within Windows Explorer, the file will be opened within Excel.

There is one caveat with respect to using Excel. Excel is limited to 1,048,576 rows of data. If you have  $m$  observations per replication and  $r$  replications, then  $m \times r + (r + 7)$  needs to be less than 1,048,576. You should use Access to process the data if you have a large amount of data, but if you do not have a lot of data, you can do the work in Excel. The analysis using both Excel and Access will be demonstrated.

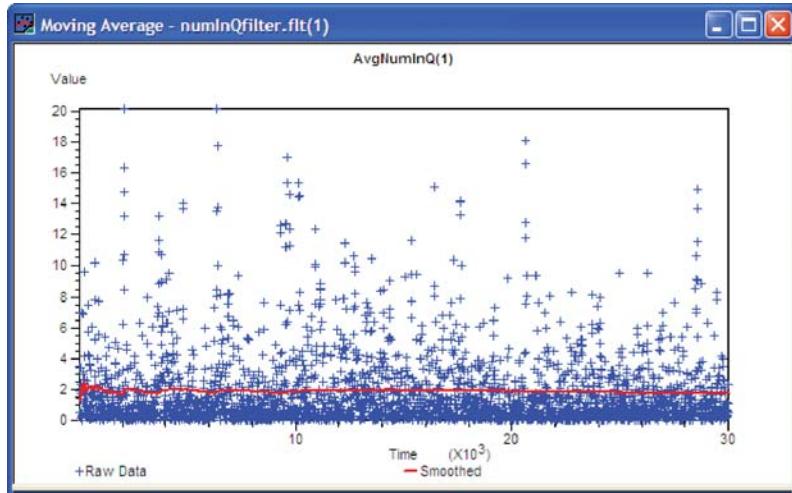


Figure 7.41 Plot of filtered time-persistent data.

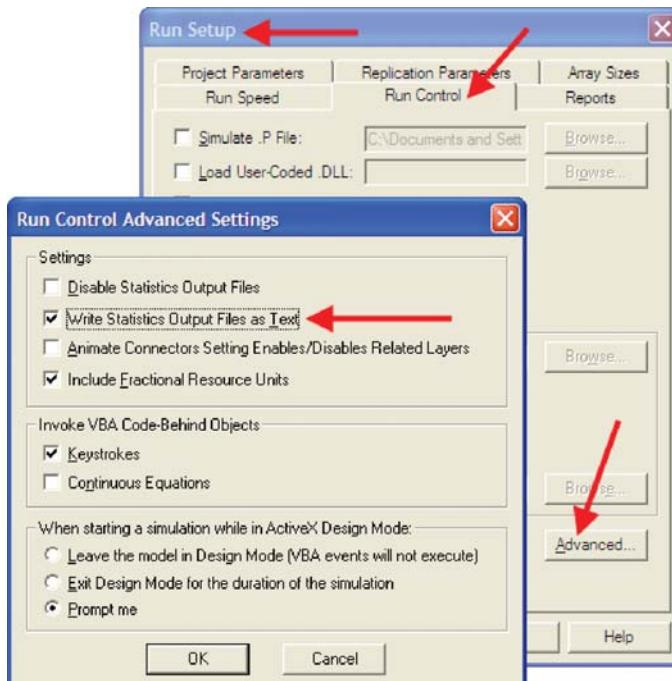


Figure 7.42 Writing statistics output files as text.

Figure 7.44 illustrates the structure of the resulting text file that is produced when using the Write Statistics Output Files as Text option. Each time a new observation is recorded (either time based or observation based), the time of the observation and the value of the observation are written to the file. When multiple replications are executed, the number (-1)

	Name	Type	Tally	Tally Output File	Expression	Report Label	Output File	...
1	AvgQTime	Tally	QTime	qtime.csv		AvgQTime		
2	AvgNumInQ	Time-Persistent	Tally 3		NQ(Seize Server.Queue)	AvgNumInQ	numinQ.csv	...

Double-click here to add a new row.

Figure 7.43 Renaming files with CSV extension.

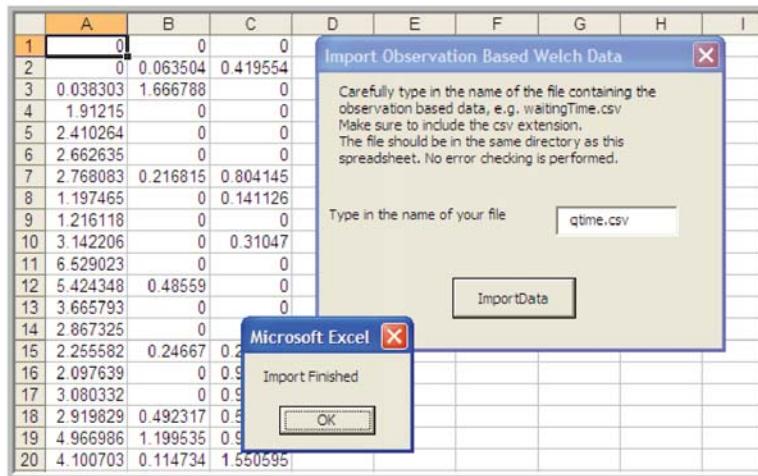
	A	B	C	D
1	Project:	Unnamed Project		
2	User:		#Replications	
3	Data item:	QTime		
4	Run date:	7/24/2006		
5	Options:	YDT		3
6				
7	Time	Observation		
8	0.346152	0		
9	0.791373	0		
10	0.889847	0.03830269		
11	2.888587	1.91214956		
12	4.178476	2.41026417		
13	6.350378	2.66263546		
9985	9998.654	1.52816907		
9986	9998.997	1.5158467		
9987	9999.233	1.46073417		End of replication indicator
9988	-1	0		
9989	0.107442	0		

Figure 7.44 Resulting CSV file as shown in Excel.

is used in the time field to indicate that a replication has ended. Using these file characteristics, software can be written to post-process the *csv* files in any manner that is required for the analysis.

To illustrate the use of Excel, the simulation was run for three replications with a run length of 10,000 minutes. This should produce about 30,000 observations of the waiting time within the queue. With these settings, you are well under the limitations imposed by Excel, but you can see that increasing the number of replications or the run length can easily exceed the spreadsheet row limitation.

Open up the file *MakeWelchPlotNew.xls* supplied with this chapter. This file contains VBA macros that will allow the importing of the observation-based and time-based data values generated from Arena™. Using Tools > Macro > Macros, open up the defined macros for the workbook and select the macro called *ShowObservationBasedWelchForm*. This will bring up an Excel VBA form for importing the data. You should then type in the name of the file containing the observations. The macro does no error checking so make sure that you type in the name of the file correctly (with the *csv* extension) and that the file is located in the same directory as the workbook. Once you press the import button, the observations from each replication will be imported into columns starting at column A of a worksheet named *WelchData* as shown in Figure 7.45.



**Figure 7.45** Importing observation data into Excel.

The other macro *ShowTimeBasedWelchForm* will allow you to import time-persistent data into Excel. When using this form, you must provide a batching interval so that the time-persistent data can be batched into equal intervals. Then the analysis is the same except that when you identify a deletion point, it will be based on the number of intervals, from which you can then compute the time of the warm-up period by multiplying the number of intervals and the length of the interval.

After the import is completed, you should scroll down through the values. You will see that the number of observations per replication is not the same. When you build the Welch plot, you will want to include only those points where you have values for each replication. In this particular example, the last row containing values for each replication is row 9980. To build the Welch plot, you need to compute the average across replications for each observation. It is also useful to compute the cumulative average for plotting it on top of the averages. The spreadsheet *MakeWelchPlotNew.xls* has the completed Welch plot on the worksheet labeled *WelchPlot*. To make this plot, the imported data was used. Then, columns were created to hold the observation index, the average across the replications, the running sum of the across replication averages, and the cumulative sum of the across replication averages as shown in Figure 7.46. Then, an Excel scatter plot of the data was made using the observation index as the *x*-axis series and the average and cumulative average for the other series of the chart. The plot indicates that after about number 4000, there is a decrease in the cumulative average and that after about 8000, there does not appear to be much change. With this data, additional analysis can be performed. In addition, if more replications are available, you will be more likely to be able to discern a pattern. VBA code was also developed for Microsoft Access to allow the processing of larger data sets. The database file *MakeWelchPlot.mdb* which accompanies this chapter can be used for analysis of larger data sets.

The processing of the data within Microsoft Access is very similar to that done in Excel except queries can be used to compute many of the quantities. To illustrate the use of Access, the simulation was reexecuted with 30 replications with a replication length of 30,000 minutes. This will generate very large files for *numInQ.csv* and *qtime.csv* (at least 22 MB of data). For convenience in what follows, these files were renamed *numInQ30R.csv*

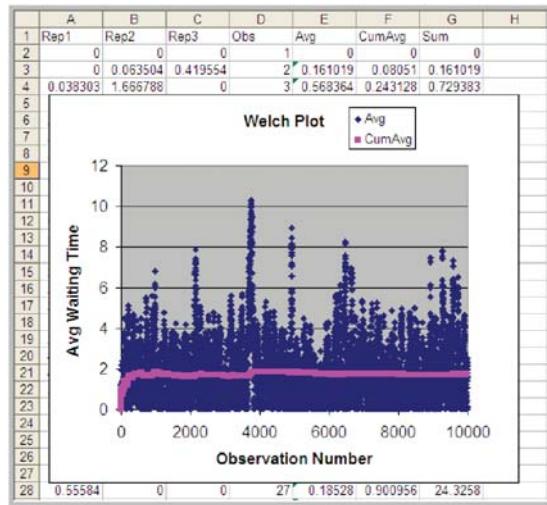


Figure 7.46 The Welch plot for Excel imported data.

Observations : Table				
TNOW	Obs	ObsNum	RepNum	
0.34615226757	0	1	1	
0.79137334054	0	2	1	
0.88984738325	0.03830268896	3	1	
2.88858671898	1.91214956146	4	1	
4.17847559005	2.41026417198	5	1	
6.35037849048	2.66263545724	6	1	
6.59692476770	2.76808311966	7	1	
6.74304639913	1.19746525635	8	1	
6.88278760814	1.21611809393	9	1	
9.43981320965	3.14220640404	10	1	

Figure 7.47 Observations table after import.

and *qtime30R.csv*. Open up the database file *MakeWelchPlot.mdb* (answer No when asked to block expressions and Yes to the unblocked expression warning). Within Access, navigate to the Forms and open up the form named *frmImportObsBased*. This form will walk you through the steps to import the data. Since the files are so large, you should expect a significant delay as the data is read in and processed. The import process will create a table called *Observations* which has the time of the observation, the observation, the observation number, and the replication associated with the observation as shown in Figure 7.47.

Selecting the *MakeWelchData* button will cause the creation of a table with the averages across the replications for each observation as shown in Figure 7.48.

You will then be offered the opportunity to mark the records within the *WelchData* table into batches. This process assigns each observation to a batch to facilitate plotting within Access. This also computes the cumulative average column. You now have everything that you need to make the Welch plots. This data can be exported to other plotting packages or plots can be made using Access's charting capabilities. Unfortunately, at the time of this writing, the size of charts within Access is limited to only 4000 data points. To collapse the data further by averaging across batches, you can use the *Make Batched Welch Data Table*

**WelchData : Table**

ObsNum	AvgOfObs	StDevOfObs	CountOfObs	BatNum	CumAvg
1	0	0	30	1	0
2	0.15846701255	0.3998252918	30	1	0.07923350627
3	0.36975412711	0.59956431398	30	1	0.17607371322
4	0.59712708297	1.00565998378	30	1	0.28133705566
5	0.73956953988	1.01445315852	30	1	0.37298355250
6	0.63968757215	0.97589803062	30	1	0.41743422244
7	0.59678078785	1.01095625657	30	1	0.44305516036
8	0.70254219975	1.38311726398	30	2	0.47549104028
9	0.70809367910	1.14586705271	30	2	0.50133577793
10	0.79829410108	1.22587062514	30	2	0.53103161024
11	0.90706994066	1.72979607558	30	2	0.56521691301
12	1.19860397882	1.95228747951	30	2	0.61799916849
13	1.09102010939	1.78415415856	30	2	0.65438539472
14	1.41848163754	2.26422836074	30	2	0.70896369778
15	1.49400043732	2.14002904459	30	3	0.76129948041

Record: [Back] [First] [Next] [Last] [Home] 1 [Next] [Last] [All] of 29569

Figure 7.48 WelchData table.

**BatchWelchDataTable : Table**

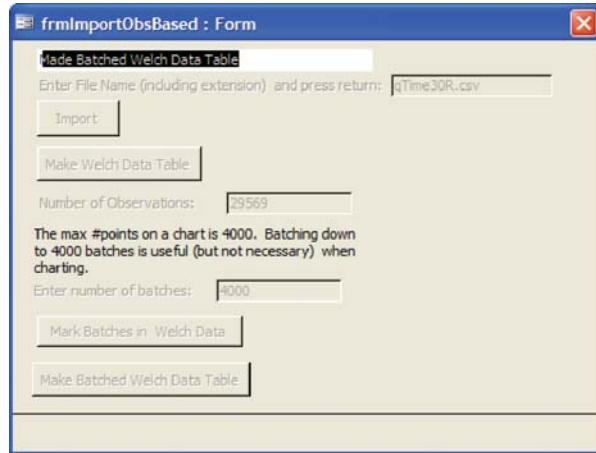
BatNum	AvgOfAvgOfObs	CumAvg
1	0.44305516036	0.44305516036
2	0.97487223519	0.70896369778
3	1.48210633283	0.96667790946
4	1.28369342375	1.04593178803
5	1.56587888426	1.14992120728
6	1.31879948855	1.17806758749
7	1.34084573780	1.20132160896
8	1.80004622952	1.27616218653
9	1.45954992211	1.2965386016
10	1.35987011345	1.30287175278
11	1.65848389806	1.33520012963
12	1.51605364164	1.35027125563
13	1.64511306143	1.37295139453
14	1.60635840576	1.38962332391
15	1.73317539592	1.41252679538

Record: [Back] [First] [Next] [Last] [Home] 1 [Next] [Last] [All] of 4225

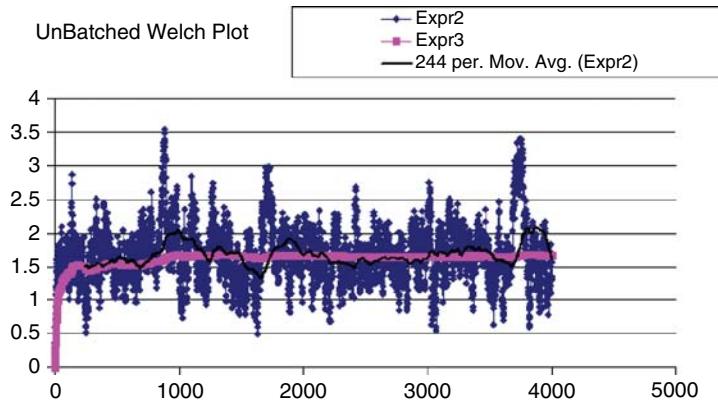
Figure 7.49 Batch averages and cumulative average.

button which takes the assigned batches and computes the average for each batch producing the BatchWelchDataTable as shown in Figure 7.49. After the whole import process has been completed, the importing form should look as shown in Figure 7.50.

The report section of the database has a number of pre-made reports that will show charts of the data as illustrated in Figure 7.51. Unfortunately, the chart size limitation of Access will truncate the plot after only 4000 observations. The figure indicates that the data is less variable because it has been averaged over 30 replications. It is also clear that it is centered around 1.6 and that at least 2000 data points are needed for the warm-up period.



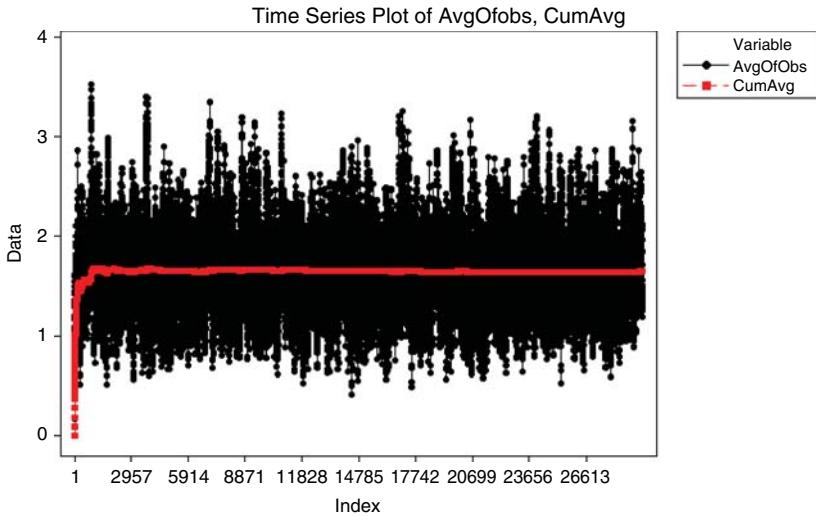
**Figure 7.50** Importing observation-based data form.



**Figure 7.51** Result of running report CumulativeAvgUnBatchedGraph.

Figure 7.52 illustrates a plot of the data from the WelchData table. This graph includes all the observations. For illustrative purposes, the analytical answer of 1.63 has been included on the graph. From this graph, it is clear that after 6000 data points, the Welch plot has settled down. Based on all this analysis, at least 6000 data points should be used for the warm-up period. To repeat the process on another data file, you must either delete or rename the tables (Observation, WelchData, BatchWelchDataTable). To reassign observations in WelchData to different batches (i.e., to rebatch the data), you can use the macro called `MarkBatchNumbersInWelchDataTable`. This macro calls a VBA function that requires the desired number of observations per batch.

Once you have performed the warm-up analysis, you still need to use your simulation model to estimate system performance. Because the process of analyzing the warm-up period involves saving the data, you could use the already saved data to estimate your system performance after truncating the initial portion of the data from the data sets. The Output Analyzer facilitates this via the Batch/Truncate dialog, which allows you to truncate the



**Figure 7.52** The Welch plot of all data.

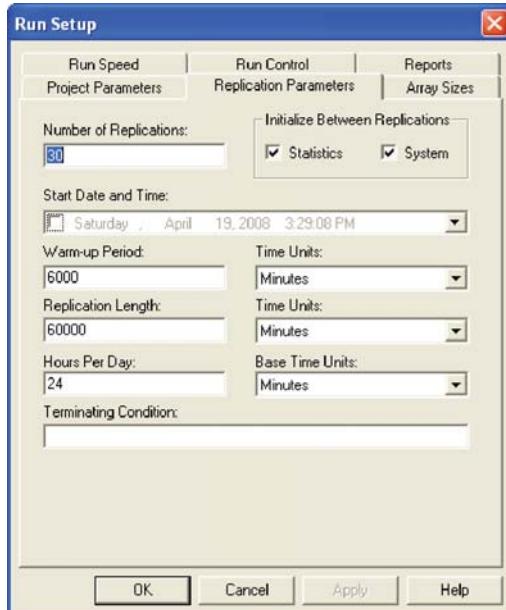
data. If the analysis was done in Excel or Access, it is simply a matter of deleting the portion of the data contained in the warm-up period. If rerunning the simulation is relatively inexpensive, then you can simply set the warm-up period in the Run Setup > Replication Parameters dialog and execute the model. Following the rule of thumb that the length of the run should be at least 10 times the warm-up period, the simulation was rerun with the settings given in Figure 7.53 (30 replications, 6000-minute warm-up period, 60,000-minute replication length). The results shown in Figure 7.54 indicate that there does not appear to be any significant bias with these replication settings.

The true waiting time in the queue is  $1.633$ , and it is clear that the 95% confidence interval contains this value.

The process described here for determining the warm-up period for steady-state simulation is tedious and time consuming. Research into automating this process is still an active area of investigation. The recent work by Robinson [2005] and Rossetti and Li [2005] holds some promise in this regard; however, there remains the need to integrate these methods into computer simulation software. Even though determining the warm-up period is tedious, some consideration of the warm-up period should be done for infinite-horizon simulations.

Once the warm-up period has been found, you can set the Warm-up Period field in the Run Setup > Replication Parameters dialog to the time of the warm up. Then, you can use the method of replication–deletion to perform your simulation experiments. Thus, all the discussion previously presented on the analysis of finite-horizon simulations can be applied.

When determining the number of replications, you can apply the fixed sample size procedure after performing a pilot run. If the analysis indicates that you need to make more runs to meet your confidence interval half-width, you have two alternatives: (i) increase the number of replications or (ii) keep the same number of replications but increase the length of each replication. If  $n_0$  was the initial number of replications and  $n$  is the number of replications recommended by the sample size determination procedure, then you can instead set  $T_e$  equal to  $(n/n_0)T_e$  and run  $n_0$  replications. Thus, you will still have approximately the same amount of data collected over your replications, but the longer run length may reduce the effect of initialization bias.



**Figure 7.53** Replication–deletion settings for the example.

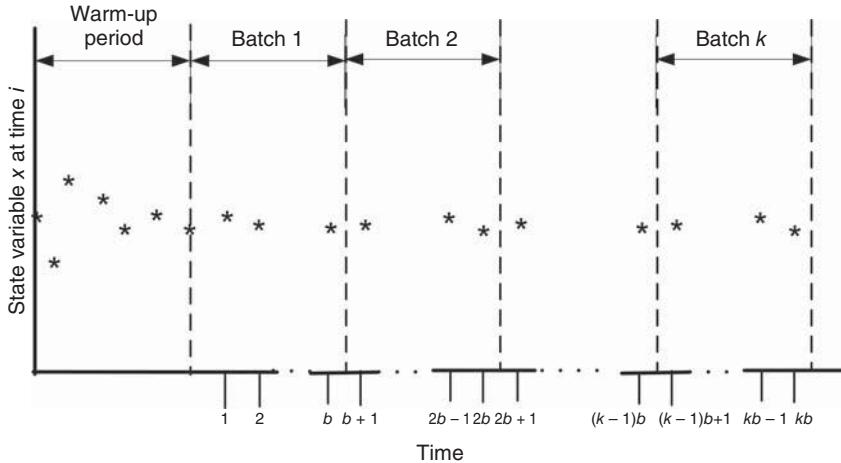
Waiting Time		
	Average	Half Width
Seize Server.Queue	1.6354	0.02
<b>Other</b>		
Number Waiting		
	Average	Half Width
Seize Server.Queue	1.6355	0.02

**Figure 7.54** Results based on 30 replications.

As previously mentioned, the method of replication–deletion causes each replication to delete the initial portion of the run. As an alternative, you can make one long run and delete the initial portion only once. When analyzing an infinite-horizon simulation based on one long replication, a method is needed to address the correlation present in the within-replication data. The method of batch means is often used in this case and has been automated in Arena™. The next section discusses the statistical basis for the batch means method and addresses some of the practical issues of using it within Arena™.

#### 7.4.4 The Method of Batch Means

In the batch mean method, only one simulation run is executed. After deleting the warm-up period, the remainder of the run is divided into  $k$  batches, with each batch average representing a single observation as illustrated in Figure 7.55.



**Figure 7.55** Illustration of the batch means method.

The advantages of the batch means method are that it entails a long simulation run, thus dampening the effect of the initial conditions. The disadvantage is that the within-replication data are correlated and unless properly formed, the batches may also exhibit a strong degree of correlation.

The following presentation assumes that a warm-up analysis has already been performed and that the data that has been collected occurs after the warm-up period. For simplicity, the presentation assumes observation-based data. The discussion also applies to time-based data that has been cut into discrete equally spaced intervals of time as described in Section 7.4.3. Therefore, assume that a series of observations  $(X_1, X_2, X_3, \dots, X_n)$  is available from within the one long replication after the warm-up period. As shown earlier at the beginning of Section 7.4, the within-replication data can be highly correlated. In that section, it was mentioned that standard confidence intervals based on

$$S^2(n) = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$$

are not appropriate for this type of data. Suppose you were to ignore the correlation, what would be the harm? In essence, a confidence interval implies a certain level of confidence in the decisions based on the confidence interval. When you use  $S^2(n)$  as defined above, you will not achieve the desired level of confidence because  $S^2(n)$  is a biased estimator for the variance of  $\bar{X}$  when the data are correlated. Under the assumption that the data are covariance stationary, an assessment of the harm in ignoring the correlation can be made. For a series that is covariance stationary, one can show that

$$\text{Var}(\bar{X}) = \frac{\gamma_0}{n} \left[ 1 + 2 \sum_{k=1}^{n-1} \left(1 - \frac{k}{n}\right) \rho_k \right] \quad (7.19)$$

where  $\gamma_0 = \text{Var}(X_i)$ ,  $\gamma_k = \text{Cov}(X_i, X_{i+k})$ , and  $\rho_k = \gamma_k/\gamma_0$  for  $k = 1, 2, \dots, n-1$ .

When the data are correlated,  $S^2/n$  is a biased estimator of  $\text{Var}(\bar{X})$ . To show this, you need to compute the expected value of  $S^2/n$  as follows:

$$\mathbb{E}[S^2/n] = \frac{\gamma_0}{n} \left[ 1 - \frac{2R}{n-1} \right] \quad (7.20)$$

where

$$R = \sum_{k=1}^{n-1} \left(1 - \frac{k}{n}\right) \rho_k \quad (7.21)$$

Bias is defined as the difference between the expected value of the estimator and the quantity being estimated. In this case, the bias can be computed with some algebra as

$$\text{Bias} = \mathbb{E}[S^2/n] - \text{Var}(\bar{Y}) = \frac{-2\gamma_0 R}{n-1} \quad (7.22)$$

Since  $\gamma_0 > 0$  and  $n > 1$ , the sign of the bias depends on the quantity  $R$  and thus on the correlation. There are three cases to consider: zero correlation, negative correlation, and positive correlation. Since  $-1 \leq \rho_k \leq 1$ , examining the limiting values for the correlation will determine the range of the bias.

For positive correlation,  $0 \leq \rho_k \leq 1$ , the bias will be negative ( $-\gamma_0 \leq \text{Bias} \leq 0$ ). Thus, the bias is negative if the correlation is positive, and the bias is positive if the correlation is negative. In the case of positive correlation,  $S^2/n$  underestimates the  $\text{Var}(\bar{X})$ . Thus, using  $S^2/n$  to form confidence intervals will make the confidence intervals too short. You will have unjustified confidence in the point estimate in this case. The true confidence will not be the desired  $1 - \alpha$ . Decisions based on positively correlated data will have a higher than planned risk of making an error based on the confidence interval.

One can easily show that for negative correlation,  $-1 \leq \rho_k \leq 0$ , the bias will be positive ( $0 \leq \text{Bias} \leq \gamma_0$ ). In the case of negatively correlated data,  $S^2/n$  overestimates the  $\text{Var}(\bar{X})$ . A confidence interval based on  $S^2/n$  will be too wide, and the true quality of the estimate will be better than indicated. The true confidence coefficient will not be the desired  $1 - \alpha$ ; it will be greater than  $1 - \alpha$ .

Of the two cases, the positively correlated case is the more severe in terms of its effect on the decision-making process; however, both are problems. Thus, the naive use of  $S^2/n$  for dependent data is highly unwarranted. If you want to build confidence intervals on  $\bar{X}$ , you need to find an unbiased estimator of the  $\text{Var}(\bar{X})$ .

The method of batch means provides a way to develop (at least approximately) an unbiased estimator for  $\text{Var}(\bar{X})$ . Assuming that you have a series of data points, the method of batch means divides the data into subsequences of contiguous batches

$$\begin{aligned} & X_1, X_2, \dots, X_b \cdots X_{b+1}, X_{b+2}, \dots, X_{2b} \cdots \\ & \underbrace{\phantom{X_1, X_2, \dots, X_b}}_{\text{batch1}} \qquad \qquad \qquad \underbrace{\phantom{X_{b+1}, X_{b+2}, \dots, X_{2b}}}_{\text{batch2}} \\ & X_{(j-1)b+1}, X_{(j-1)b+2}, \dots, X_{jb} \cdots X_{(k-1)b+1}, X_{(k-1)b+2}, \dots, X_{kb} \\ & \underbrace{\phantom{X_{(j-1)b+1}, X_{(j-1)b+2}, \dots, X_{jb}}}_{\text{batch}j} \qquad \qquad \qquad \underbrace{\phantom{X_{(k-1)b+1}, X_{(k-1)b+2}, \dots, X_{kb}}}_{\text{batch}k} \end{aligned}$$

and computes the sample average of the batches. Let  $k$  be the number of batches each consisting of  $b$  observations, so that  $k = \lfloor n/b \rfloor$ . If  $b$  is not a divisor of  $n$ , then the last

$(n - kb)$  data points will not be used. Define  $\bar{X}_j(b)$  as the  $j$ th batch mean for  $j = 1, 2, \dots, k$ , where

$$\bar{X}_j(b) = \frac{1}{b} \sum_{i=1}^b X_{(j-1)b+i} \quad (7.23)$$

Each of the batch means are treated like observations in the batch means series. For example, if the batch means are relabeled as  $Y_j = \bar{X}_j(b)$ , the batching process simply produces another series of data  $(Y_1, Y_2, Y_3, \dots, Y_k)$ , which may be more like a random sample. To form a  $(1 - \alpha)\%$  confidence interval, you simply treat this new series like a random sample and compute approximate confidence intervals using the sample average and sample variance of the batch means series:

$$\bar{Y}(k) = \frac{1}{k} \sum_{j=1}^k Y_j \quad (7.24)$$

$$S_b^2(k) = \frac{1}{k-1} \sum_{j=1}^k (Y_j - \bar{Y})^2 \quad (7.25)$$

$$\bar{Y}(k) \pm t_{\alpha/2, k-1} \frac{S_b(k)}{\sqrt{k}} \quad (7.26)$$

Since the original  $X$ 's are covariance stationary, it follows that the resulting batch means are also covariance stationary. One can show, see Alexopoulos and Seila [1998], that the correlation in the batch means reduces as both the size of the batches,  $b$ , and the number of data points,  $n$ , increases. In addition, one can show that  $S_b^2(k)/k$  approximates  $\text{Var}(\bar{X})$  with error that reduces as both  $b$  and  $n$  increase toward infinity.

The basic difficulty with the batch means method is determining the batch size or alternatively the number of batches. Larger batch sizes are good for independence but reduce the number of batches, resulting in higher variance for the estimator. Schmeiser [1982] performed an analysis that suggests that there is little benefit if the number of batches is larger than 30 and recommended that the number of batches remain in the range from 10 to 30. However, when trying to access whether or not the batches are independent, it is better to have a large number of batches ( $>100$ ) so that tests on the lag- $k$  correlation have better statistical properties.

There are a variety of procedures that will automatically batch the data as it is collected have been developed, see, for example, Fishman and Yarberry [1997], Steiger and Wilson [2002], and Banks et al. [2005]. Arena™ has its own batching algorithm. The batching algorithm is described in Kelton et al. [2004], p. 311. See also Fishman [2001], p. 254 for an analysis of the effectiveness of the algorithm.

The discussion here is based on the description in Kelton et al. [2004]. When the algorithm has recorded a sufficient amount of data, it begins by forming  $k = 20$  batches. As more data is collected, additional batches are formed until  $k = 40$  batches are collected. When 40 batches are formed, the algorithm collapses the number of batches back to 20 by averaging each pair of batches. This has the net effect of doubling the batch size. This process is repeated as more data is collected, thereby ensuring that the number of batches is between 20 and 39. The algorithm begins the formation of batches when it has at least 320 observations of tally-based data.

For time-persistent data, the algorithm requires that there were at least 5 time units during which the time-based variable changed 320 times. If there are not enough observations within a run, then *Insufficient* is reported for the half-width value on the output reports. In addition, the algorithm also tests to see if the lag-1 correlation is significant by testing the hypothesis that the batch means are uncorrelated using the following test statistic, see Alexopoulos and Seila [1998]:

$$C = \sqrt{\frac{k^2 - 1}{k - 2} \left[ \hat{\rho}_1 + \frac{[Y_1 - \bar{Y}]^2 + [Y_k - \bar{Y}]^2}{2 \sum_{j=1}^k (Y_j - \bar{Y})^2} \right]} \quad (7.27)$$

$$\hat{\rho}_1 = \frac{\sum_{j=1}^{k-1} (Y_j - \bar{Y})(Y_{j+1} - \bar{Y})}{\sum_{j=1}^k (Y_j - \bar{Y})^2} \quad (7.28)$$

The hypothesis is rejected if  $C > z_\alpha$  for a given confidence level  $\alpha$ . If the batch means do not pass the test, *Correlated* is reported for the half-width on the statistical reports.

#### 7.4.5 Performing the Method of Batch Means

Performing the method of batch means in Arena™ is relatively straightforward. The following assumes that a warm-up period analysis has already been performed. Since batches are formed during the simulation run and the confidence intervals are based on the batches, the primary concern will be to determine the run length that will ensure a desired half-width on the confidence intervals. A fixed-sampling-based method and a sequential sampling method will be illustrated.

The analysis performed to determine the warm-up period should give you some information concerning how long to make this single run and how long to set its warm-up period. Assume that a warm-up analysis has been performed using  $n_0$  replications of length  $T_e$  and that the analysis has indicated a warm-up period of length  $T_w$ .

As previously discussed, the method of replication deletion spreads the risk of choosing a bad initial condition across multiple replications. The method of batch means relies on only one replication. If you were satisfied with the warm-up period analysis based on  $n_0$  replications and you were going to perform replication deletion, then you are willing to throw away the observations contained in at least  $n_0 \times T_w$  time units and you are willing to use the data collected over  $n_0 \times (T_e - T_w)$  time units. Therefore, the warm-up period for the single replication can be set at  $n_0 \times T_w$  and the run length can be set at  $n_0 \times T_e$ .

For example, suppose your warm-up analysis was based on the initial results shown in Section 7.4.3, that is,  $n_0 = 10$ ,  $T_e = 30,000$ ,  $T_w = 3000$ . Thus, your starting run length would be  $n_0 \times T_e = 10 \times 30,000 = 300,000$  and the warm-up period will be  $n_0 \times T_w = 30,000$ . For these setting, the results shown in Figure 7.56 are very close to the results for the replication-deletion example.

Suppose now you want to ensure that the half-widths from a single replication are less than a given error bound. The half-widths reported by the simulation for a single replication are based on the *batch means*. You can get an approximate idea of how much to increase the length of the replication by using the functions: TNUMBAT(Tally ID) and TBATSIZ(Tally ID) for observation-based statistics or DNUMBAT(DSTAT ID) and DBATSIZ(DSTAT ID) in conjunction with the half-width sample size determination formula.

$$n \cong n_0 \left( \frac{h_0}{h} \right)^2$$

Waiting Time		Average	Half Width
Seize Server.Queue		1.6225	0.061752921
<b>Other</b>			
Number Waiting		Average	Half Width
Seize Server.Queue		1.6241	0.064281088

Figure 7.56 Initial batch means results.

Statistic - Advanced Process				
	Name	Type	Expression	Report Label
1	NBforQT	Output	TNUMBAT(QTime)	NBforQT
2	BSforQT	Output	TBATSIZ(QTime)	BSforQT
Double-click here to add a new row.				

Figure 7.57 OUTPUT statistics to get number of batches and batch size.

Output		Value
BSforQT		8192.00
NBforQT		32.0000

Figure 7.58 Results for number of batches and batch Size

In this case, you interpret  $n$  and  $n_0$  as the number of batches. OUTPUT statistics can be added to the model to observe the number of batches for the waiting time in queue and for the size of each batch, as shown in Figure 7.57. The resulting values for the number of batches formed for the waiting times and the size of the batches are given in Figure 7.58. Using this information in the half-width-based sample size formula with  $n_0 = 32$ ,  $h_0 = 0.06$ , and  $h = 0.02$  yields

$$n \cong n_0 \frac{h_0^2}{h^2} = 32 \times \frac{(0.06)^2}{(0.02)^2} = 288 \text{ batches}$$

Since each batch in the run had 8192 observations, this yields the need for additional observations for the waiting time in the queue. Since, in this model, customers arrive at a mean rate of 1 per minute, this requires about 2,359,296 additional time units of simulation. Because of the warm-up period, you, therefore, need to set  $T_e$  equal to  $(2,359,296 + 30,000 = 2,389,296)$ . Rerunning the simulation yields the results shown in Figure 7.59. The results show that the half-width meets the desired criteria. This approach is approximate since you do not know how the observations will be batched when making the final run.

Rather than trying to fix the amount of sampling, you might instead try to use a sequential sampling technique that is based on the half-width computed during the simulation run. This

Waiting Time		Average	Half Width
Seize Server.Queue		1.6403	0.017292365
<b>Other</b>			
Number Waiting		Average	Half Width
Seize Server.Queue		1.6422	0.018253294

Figure 7.59 Batch means results for fixed sample size.

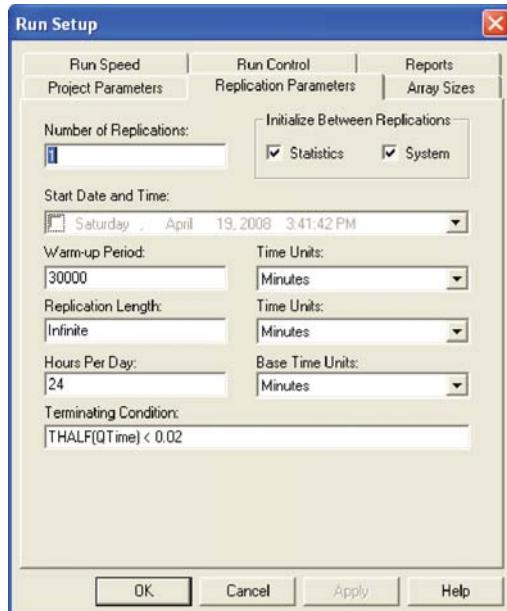


Figure 7.60 Sequential sampling using terminating condition.

is easy to do by supplying the appropriate expression within the Terminating Condition field on the Run Setup > Replication Parameters dialog.

Figure 7.60 illustrates that you can use a Boolean expression within the Terminating Condition field. In this case, the THALF(Tally ID) function is used to specify that the simulation should terminate when the half-width criteria is met. The batching algorithm computes the value of THALF(Tally ID) after sufficient data has been observed. This expression can be expanded to include other performance measures in a compound Boolean statement.

The results of running the simulation based on the sequential method are given in Figure 7.61. In this case, the simulation run ended at approximately time 1,928,385. This is lower than the time specified for the fixed sampling procedure (but the difference is not excessive).

Once the warm-up period has been analyzed, performing infinite-horizon simulations using the batch means method is relatively straightforward. A disadvantage of this method

Waiting Time	Average	Half Width
Seize Server.Queue	1.6366	0.019929123
<b>Other</b>		
Number Waiting	Average	Half Width
Seize Server.Queue	1.6384	0.021162914

**Figure 7.61** Results for infinite-horizon sequential sampling method.

is that it will be more difficult to use the statistical methods available within the Process Analyzer or within OptQuest because they assume a replication–deletion approach. If you are faced with an infinite-horizon simulation, then you can use either the replication–deletion approach or the batch means method readily from within Arena™. In either case, you should investigate if there may be any problems related to initialization bias. If you use the replication–deletion approach, you should play it safe when specifying the warm-up period. Making the warm-up period longer than you think, it should be better than replicating a poor choice. When performing an infinite-horizon simulation based on one long run, you should make sure that your run length is long enough. A long-run length can help to “wash out” the effects of initial condition bias.

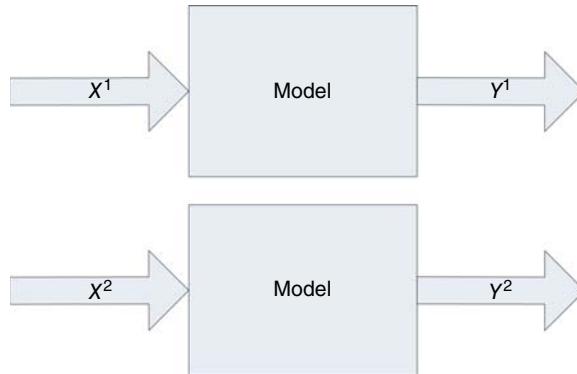
Ideally, in the situation where you have to make many simulation experiments using different parameter settings of the same model, you should perform a warm-up analysis for each design configuration. In practice, this is not readily feasible when there are a large number of experiments. In this situation, you should use your common sense to pick the design configurations (and performance measures) that you feel will most likely suffer from initialization bias. If you can determine long enough warm-up periods for these configurations, the other configurations should be relatively safe from the problem by using the longest warm-up period found.

There are a number of other techniques that have been developed for the analysis of infinite-horizon simulations including the standardized time series method, the regenerative method, and spectral methods. An overview of these methods and others can be found in Alexopoulos and Seila [1998] and Law [2007].

So far you have learned how to analyze the data from one design configuration. A key use of simulation is to be able to compare alternative system configurations and to assist in choosing which configurations are best according to the decision criteria. The next section presents how to use simulation and the tools available within Arena™ to compare different system configurations.

## 7.5 COMPARING SYSTEM CONFIGURATIONS

The previous sections have concentrated on estimating the performance of a system through the execution of a single simulation model. The running of the model requires the specification of the input variables (e.g., mean time between arrivals and service distribution) and the structure of the model (e.g., FIFO queue and process flow). The specification of a set of inputs (variables and/or structure) represents a particular system configuration, which is then simulated to estimate performance. To be able to simulate design configurations, you may have to build different models or you may be able to use the same model supplied with



**Figure 7.62** Multiple inputs on models represent different system configurations.

different values of the program inputs. In either situation, you now have different design configurations that can be compared. This allows the performance of the system to be estimated under a wide variety of controlled conditions. It is this ability to easily perform these what-if simulations that make simulation such a useful analysis tool. Figure 7.62 represents the notion of using different inputs to get different outputs.

Naturally, when you have different design configurations, you would like to know which configurations are better than the others. Since the simulations are driven by random variables, the outputs from each configuration (e.g.,  $Y^1$  and  $Y^2$ ) are also random variables. The estimate of the performance of each system must be analyzed using statistical methods to ensure that the differences in performance are not simply due to sampling error. In other words, you want to be confident that one system is statistically better (or worse) than the other system.

### 7.5.1 Comparing Two Systems

The techniques for comparing two systems via simulation are essentially the same as that found in books that cover the statistical analysis of two samples (e.g., Montgomery and Runger, [2006]). This section begins with a review of these methods. Assume that samples from two different populations (system configurations) are available:

$X_{11}, X_{12}, \dots, X_{1n_1}$  a sample of size  $n_1$  from system configuration 1

$X_{21}, X_{22}, \dots, X_{2n_2}$  a sample of size  $n_2$  from system configuration 2

The samples represent a performance measure of the system that will be used in a decision regarding which system configuration is preferred. For example, the performance measure may be the average system throughput per day, and you want to pick the design configuration that has highest throughput.

Assume that each system configuration has an unknown population mean for the performance measure of interest,  $E[X_1] = \theta_1$  and  $E[X_2] = \theta_2$ . Thus, the problem is to determine, with some statistical confidence, whether  $\theta_1 < \theta_2$  or, alternatively,  $\theta_1 > \theta_2$ . Since the system configurations are different, an analysis of the situation of whether  $\theta_1 = \theta_2$  is of less relevance in this context.

Define  $\theta = \theta_1 - \theta_2$  as the mean difference in performance between the two systems. Clearly, if you can determine whether  $\theta > 0$  or  $\theta < 0$ , you can determine whether  $\theta_1 < \theta_2$  or  $\theta_1 > \theta_2$ . Thus, it is sufficient to concentrate on the difference in performance between the two systems.

Given samples from two different populations, there are a number of ways in which the analysis can proceed based on different assumptions concerning the samples. The first common assumption is that the observations within each sample for each configuration form a random sample. That is, the samples represent independent and identically distributed random variables. Within the context of simulation, this can be easily achieved for a given system configuration by performing replications. For example, this means that  $X_{11}, X_{12}, \dots, X_{1n_1}$  are the observations from  $n_1$  replications of the first system configuration. A second common assumption is that both populations are normally distributed or that the central limit theorem can be used so that sample averages are at least approximately normal.

To proceed with further analysis, assumptions concerning the population variances must be made. Many statistics textbooks present results for the case of the population variance being known. In general, this is not the case within simulation contexts, so the assumption here will be that the variances associated with the two populations are unknown. Textbooks also present cases where it is assumed that the population variances are equal. Rather than making that assumption, it is better to test a hypothesis regarding equality of population variances.

The last assumption concerns whether or not the two samples can be considered independent of each other. This last assumption is very important within the context of simulation. Unless you take specific actions to ensure that the samples will be independent, they will, in fact, be dependent because of how simulations use (reuse) the same random number streams. The possible dependence between the two samples is not necessarily a bad thing. In fact, under certain circumstance, it can be a good thing.

The following sections first present the methods for analyzing the case of unknown variance with independent samples. Then, we focus on the case of dependence between the samples. Finally, how to use Arena™ to do the work of the analysis will be illustrated.

**7.5.1.1 Analyzing Two Independent Samples** Although the variances are unknown, the unknown variances are either equal or not equal. In the situation where the variances are equal, the observations can be pooled when developing an estimate for the variance. In fact, rather than just assuming equal or not equal variances, you can (and should) use an  $F$ -test to test for the equality of variance. The  $F$ -test can be found in most elementary probability and statistics books (see Montgomery and Runger, [2006]).

The decision regarding whether  $\theta_1 < \theta_2$  can be addressed by forming confidence intervals on  $\theta = \theta_1 - \theta_2$ . Let  $\bar{X}_1, \bar{X}_2, S_1^2$ , and  $S_2^2$  be the sample averages and sample variances based on the two samples ( $k = 1, 2$ ):

$$\bar{X}_k = \frac{1}{n_k} \sum_{j=1}^{n_k} X_{kj} \quad (7.29)$$

$$S_k^2 = \frac{1}{n_k - 1} \sum_{j=1}^{n_k} (X_{kj} - \bar{X}_k)^2 \quad (7.30)$$

An estimate of  $\theta = \theta_1 - \theta_2$  is desired. This can be achieved by estimating the difference with  $\hat{D} = \bar{X}_1 - \bar{X}_2$ . To form confidence intervals on  $\hat{D} = \bar{X}_1 - \bar{X}_2$ , an estimator for the variance of  $\hat{D} = \bar{X}_1 - \bar{X}_2$  is required. Because the samples are independent, the computation of the variance of the difference is

$$\text{Var}(\hat{D}) = \text{Var}(\bar{X}_1 - \bar{X}_2) = \frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2} \quad (7.31)$$

where  $\sigma_1^2$  and  $\sigma_2^2$  are the unknown population variances. Under the assumption of equal variance,  $\sigma_1^2 = \sigma_2^2 = \sigma^2$ , this can be written as

$$\text{Var}(\hat{D}) = \text{Var}(\bar{X}_1 - \bar{X}_2) = \frac{\sigma^2}{n_1} + \frac{\sigma^2}{n_2} = \sigma^2 \left( \frac{1}{n_1} + \frac{1}{n_2} \right) \quad (7.32)$$

where  $\sigma^2$  is the common unknown variance. A pooled estimator of  $\sigma^2$  can be defined as

$$S_p^2 = \frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2} \quad (7.33)$$

Thus, a  $(1 - \alpha)\%$  confidence interval on  $\theta = \theta_1 - \theta_2$  is

$$\hat{D} \pm t_{\alpha/2, v} S_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}} \quad (7.34)$$

where  $v = n_1 + n_2 - 2$ . For the case of unequal variances, an approximate  $(1 - \alpha)\%$  confidence interval on  $\theta = \theta_1 - \theta_2$  is given by

$$\hat{D} \pm t_{\alpha/2, v} \sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}} \quad (7.35)$$

where

$$v = \left\lfloor \frac{\left( \frac{S_1^2}{n_1} + \frac{S_2^2}{n_2} \right)^2}{\frac{(S_1^2/n_1)^2}{n_1+1} + \frac{(S_2^2/n_2)^2}{n_2+1}} - 2 \right\rfloor \quad (7.36)$$

Let  $[l, u]$  be the resulting confidence interval where  $l$  and  $u$  represent the lower and upper limits of the interval by construction  $l < u$ . Thus, if  $u < 0$ , you can conclude with  $(1 - \alpha)\%$  confidence that  $\theta = \theta_1 - \theta_2 < 0$  (i.e.,  $\theta_1 < \theta_2$ ). If  $l > 0$ , you can conclude with  $(1 - \alpha)\%$  that  $\theta = \theta_1 - \theta_2 > 0$  (i.e.,  $\theta_1 > \theta_2$ ). If  $[l, u]$  contains 0, then no conclusion can be made at the given sample sizes about which system is better. This does not indicate that the system performance is the same for the two systems. You *know* that the systems are different. Thus, their performance will be different. This only indicates that you have not taken enough samples to detect the true difference. If sampling is relatively cheap, then you may want to take additional samples in order to discern an ordering between the systems.

### EXAMPLE 7.3 Testing Independent Configurations

Two configurations are under consideration for the design of an airport security checkpoint. A simulation model of each design was made. The replication values of the throughput per minute for the security station for each design are provided in the following table.

	Design 1	Design 2
1	10.98	8.93
2	8.87	9.82
3	10.53	9.27
4	9.40	8.50
5	10.10	9.63
6	10.28	9.55
7	8.86	9.30
8	9.00	9.31
9	9.99	9.23
10	9.57	8.88
11		8.05
12		8.74
$\bar{x}$	9.76	9.10
$s$	0.74	0.50
$n$	10	12

Assume that the two simulations were run independently of each other, using different random numbers. Recommend the better design with 95% confidence. According to the results,

$$\hat{D} = \bar{X}_1 - \bar{X}_2 = 9.76 - 9.1 = 0.66$$

In addition, we should test if the variances of the samples are equal. This requires an  $F$  test, with  $H_0 : \sigma_1^2 = \sigma_2^2$  versus  $H_1 : \sigma_1^2 \neq \sigma_2^2$ . Based on elementary statistics, the test statistic is  $F_0 = S_1^2/S_2^2$ . The rejection criterion is to reject  $H_0$  if  $F_0 > f_{\alpha/2,n_1-1,n_2-1}$  or  $F_0 < f_{1-\alpha/2,n_1-1,n_2-1}$ , where  $f_{p,u,v}$  is the upper percentage point of the  $F$  distribution. Assuming a 0.01 significance level for the  $F$  test, we have  $F_0 = (0.74)^2/(0.50)^2 = 2.12$ . Since  $f_{0.005,9,11} = 5.54$  and  $f_{0.995,9,11} = 0.168$ , there is not enough evidence to conclude that the variances are different at the 0.01 significance level. The value of  $f_{p,u,v}$  can be determined in Excel™ as F.INV.RT( $p$ ,  $u$ ,  $v$ ). Note also that  $f_{1-p,u,v} = 1/f_{p,v,u}$ . In R, the formula is  $f_{p,u,v} = qt(1 - p, u, v)$ , since R provides the quantile function not the upper right tail function.

Since the variances can be assumed equal, we can use the pooled variance, which is

$$\begin{aligned} S_p^2 &= \frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2} \\ &= \frac{(10 - 1)(0.74)^2 + (12 - 1)(0.5)^2}{12 + 10 - 2} \\ &= 0.384 \end{aligned}$$

Thus, a  $(1 - 0.05)\%$  confidence interval on  $\theta = \theta_1 - \theta_2$  is

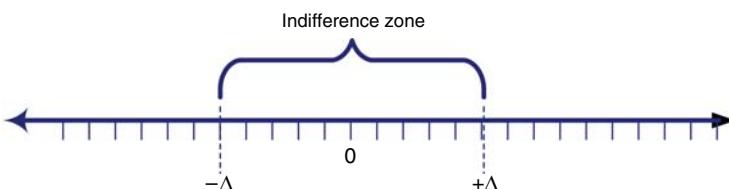
$$\begin{aligned}\hat{D} &\pm t_{\alpha/2,v} s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}} \\ 0.66 &\pm t_{0.025,20} (\sqrt{0.384}) \sqrt{\frac{1}{10} + \frac{1}{12}} \\ 0.66 &\pm (2.086)(0.6196)(0.428) \\ 0.66 &\pm 0.553\end{aligned}$$

where  $v = n_1 + n_2 - 2 = 10 + 12 - 2 = 20$ . Since this results in an interval  $[0.10, 1.21]$  that does not contain zero, we can conclude that design 1 has the higher throughput with 95% confidence.

The confidence interval can assist in making decisions regarding relative performance of the systems from a *statistically significant* standpoint. However, if you make a conclusion about the ordering of the system, it still may not be practically significant. That is, the difference in the system performance is statistically significant but the actual difference is of no practical use. For example, suppose you compare two systems in terms of throughput with resulting output  $\bar{X}_1 = 5.2$  and  $\bar{X}_2 = 5.0$  with the difference statistically significant. While the difference of 0.2 may be statistically significant, you might not be able to achieve this in the actual system. After all, you are making a decision based on a *model of the system* not on the real system. If the costs of the two systems are significantly different, you should prefer the cheaper of the two systems since there is no practical difference between the two systems. The fidelity of the difference is dependent on your modeling assumptions. Other modeling assumptions may overshadow such a small difference.

The notion of practical significance is model and performance measure dependent. One way to characterize the notion of practical significance is to conceptualize a zone of performance for which you are indifferent between the two systems. Figure 7.63 illustrates the concept of an indifference zone around the difference between the two systems. If the difference between the two systems falls in this zone, you are indifferent between the two systems (i.e. there is no practical difference).

Using the indifference zone to model the notion of practical significance, if  $u < -\Delta$ , you can conclude confidence that  $\theta_1 < \theta_2$ , and if  $l > \Delta$ , you can conclude with confidence that  $\theta_1 > \theta_2$ . If  $l$  falls within the indifference zone and  $u$  does not (or vice versa), then there is not enough evidence to make a confident conclusion. If  $[l, u]$  is totally contained within the indifference zone, then you can conclude with confidence that there is no practical difference between the two systems.



**Figure 7.63** Indifference zone.

**7.5.1.2 Analyzing Two Dependent Samples** In this situation, continue to assume that the observations within a sample are independent and identically distributed random variables; however, the samples themselves are not independent. That is, assume that  $(X_{11}, X_{12}, \dots, X_{1n_1})$  and  $(X_{21}, X_{22}, \dots, X_{2n_2})$  from the two systems are dependent. For simplicity, suppose that the difference in the configurations can be implemented using a simple parameter change within the model. For example, the mean processing time is different for the two configurations. First, run the model to produce  $(X_{11}, X_{12}, \dots, X_{1n_1})$  for configuration 1. Then, change the parameter and reexecute the model to produce  $(X_{21}, X_{22}, \dots, X_{2n_2})$  for configuration 2.

Assuming that you did nothing with respect to the random number streams, the second configuration used the same random numbers that the first configuration used. Thus, the generated responses will be correlated (dependent). In this situation, it is convenient to assume that each system is run for the same number of replications, that is,  $n_1 = n_2 = n$ . Since each replication for the two systems uses the same random number streams, the correlation between  $(X_{1j}, X_{2j})$  will not be zero; however, each pair will still be independent *across* the replications. The basic approach to analyze this situation is to compute the difference for each pair:

$$D_j = X_{1j} - X_{2j} \quad \text{for } j = 1, 2, \dots, n \quad (7.37)$$

The  $(D_1, D_2, \dots, D_n)$  will form a random sample, which can be analyzed via traditional methods. Thus, a  $(1 - \alpha)\%$  confidence interval on  $\theta = \theta_1 - \theta_2$  is

$$\bar{D} = \frac{1}{n} \sum_{j=1}^n D_j \quad (7.38)$$

$$S_D^2 = \frac{1}{n-1} \sum_{j=1}^n (D_j - \bar{D})^2 \quad (7.39)$$

$$\bar{D} \pm t_{\alpha/2, n-1} \frac{S_D}{\sqrt{n}} \quad (7.40)$$

The interpretation of the resulting confidence interval  $[l, u]$  is the same as in the independent sample approach. This is the paired-*t* confidence interval presented in statistics textbooks.

#### EXAMPLE 7.4 Testing Dependent Configurations

Two designs are under consideration for the improvement of a manufacturing system. A simulation model of each design was made. The replication values of the expected cost per day for each design are provided in the following table.

	Design 1	Design 2	Difference
1	52.56	49.92	2.64
2	48.77	47.08	1.69
3	53.49	50.62	2.87
4	50.60	48.45	2.15
5	51.60	49.20	2.40
6	51.77	49.33	2.44
7	51.09	48.81	2.27
8	53.51	50.64	2.88
9	47.44	46.08	1.36
10	47.94	46.45	1.48
$\bar{x}$	50.88	48.66	2.22
$s$	2.18	1.64	0.55
$n$	10	10	10

Assume that the two simulations were run independently using common random numbers. Recommend the better design with 95% confidence.

According to the results,

$$\bar{D} = \bar{X}_1 - \bar{X}_2 = 50.88 - 48.66 = 2.22$$

Also, we have that  $S_D^2 = (0.55)^2$ . Thus, a  $(1 - 0.05)\%$  confidence interval on  $\theta = \theta_1 - \theta_2$  is

$$\begin{aligned} \hat{D} &\pm t_{\alpha/2,n-1} \frac{S_D}{\sqrt{n}} \\ 2.22 &\pm t_{0.025,9} \frac{0.55}{\sqrt{10}} \\ 2.22 &\pm (2.261)(0.1739) \\ 2.22 &\pm 0.393 \end{aligned}$$

Since this results in an interval  $[1.827, 2.613]$  that does not contain zero, we can conclude that design 1 has the higher cost with 95% confidence.

Of the two approaches (independent vs dependent samples), the latter is much more prevalent in simulation contexts. The approach is called the method of *common random numbers (CRN)* and is a natural by-product of how most simulation languages handle their assignment of random number streams.

To understand why this method is the preferred method for comparing two systems, you need to understand the method's effect on the variance of the estimator. In the case of independent samples, the estimator of performance was  $\hat{D} = \bar{X}_1 - \bar{X}_2$ . Since

$$\begin{aligned} \bar{D} &= \frac{1}{n} \sum_{j=1}^n D_j \\ &= \frac{1}{n} \sum_{j=1}^n (X_{1j} - X_{2j}) \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{n} \sum_{j=1}^n X_{1j} - \frac{1}{n} \sum_{j=1}^n X_{2j} \\
&= \bar{X}_1 - \bar{X}_2 \\
&= \hat{D}
\end{aligned}$$

The two estimators are the same, when  $n_1 = n_2 = n$ ; however, their variances are not the same. Under the assumption of independence, computing the variance of the estimator yields

$$V_{\text{IND}} = \text{Var}(\bar{X}_1 - \bar{X}_2) = \frac{\sigma_1^2}{n} + \frac{\sigma_2^2}{n} \quad (7.41)$$

Under the assumption that the samples are not independent, the variance of the estimator is

$$V_{\text{CRN}} = \text{Var}(\bar{X}_1 - \bar{X}_2) = \frac{\sigma_1^2}{n} + \frac{\sigma_2^2}{n} - 2 \text{cov}(\bar{X}_1, \bar{X}_2) \quad (7.42)$$

If you define  $\rho_{12} = \text{corr}(\bar{X}_1, \bar{X}_2)$ , the variance for the common random number situation is

$$V_{\text{CRN}} = V_{\text{IND}} - 2\sigma_1\sigma_2\rho_{12} \quad (7.43)$$

Therefore, whenever there is positive correlation  $\rho_{12} > 0$  within the pairs, we have that  $V_{\text{CRN}} < V_{\text{IND}}$ .

If the variance of the estimator in the case of common random numbers is smaller than the variance of the estimator under independent sampling, then a *variance reduction* has been achieved. The method of common random numbers is called a variance reduction technique. If the variance reduction results in a confidence interval for  $\theta$  that is tighter than the independent case, the use of common random numbers should be preferred. The variance reduction needs to be big enough to overcome any loss in the number of degrees of freedom caused by the pairing. When the number of replications is relatively large ( $n > 30$ ), this will generally be the case since the student-*t* value does not vary appreciatively for large degrees of freedom. Notice that the method of common random numbers might backfire and cause a variance increase if there is negative correlation between the pairs. An overview of the conditions under which common random numbers may work is given in Law [2007].

This notion of pairing the outputs from each replication for the two-system configurations makes common sense. When trying to discern a difference, you want the two systems to experience the same randomness so that you can more readily infer that any difference in performance is due to the inherent difference between the systems and not caused by the random numbers.

In experimental design settings, this is called blocking on a factor. For example, if you wanted to perform and experiment to determine whether a change in a work method was better than the old method, you should use the same worker to execute both methods. If, instead, you had different workers to execute the methods, you would not be sure if any difference was due to the workers or to the proposed change in the method. In this context, the worker is the factor that should be blocked. In the simulation context, the random numbers are being blocked when using common random numbers.

**7.5.1.3 Using Common Random Numbers** The following explores how independent sampling and common random numbers can be implemented. This example returns to the LOTR Makers system. This example also illustrates the use of resources sets.

### EXAMPLE 7.5 Overtime Analysis for the LOTR Makers System

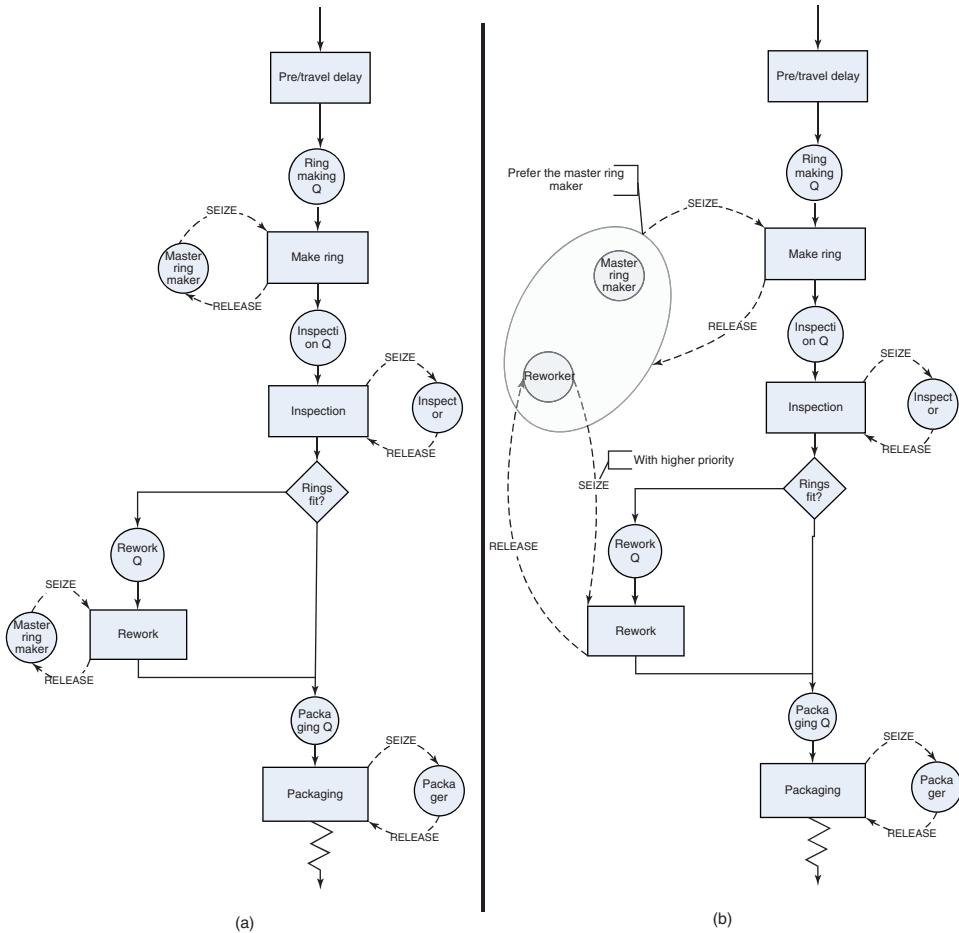
The company is interested in reducing the amount of overtime. They have noticed that a bottleneck forms at the ring making station early in the production release and that the rework station is not sufficiently utilized. Thus, they would like to test the sharing of the rework worker between the two stations. The rework worker should be assigned to both the rework station and the ring making station. If a pair of rings arrives to the ring making station either the master ring maker or the rework worker can make the rings. If both ring makers are available, the master ring maker should be preferred. If a pair of rings needs rework, the rework worker should be given priority to the rework. The rework worker is not as skilled as the master ring maker and the time to make the rings varies depending on the maker. The master ring maker still takes a UNIF(5,15) minutes to make rings; however, the shared rework worker has a much more variable process time. The rework worker can make rings in 30 minutes according to an exponential distribution. In addition, the rework worker must walk from station to station to perform the tasks and needs a bit more time to change from one task to the next. It is estimated that it will take the rework work an extra 10 minutes per task. Thus, the rework worker's ring making time is on an average 10 + EXPO(30) minutes, and the rework worker's time to process rework is now 15+WEIB(15,3) minutes.

In addition to the sharing of the rework craftsman, management has noted that the release of the jobs at the beginning of the day is not well represented by the previous model. In fact, after the jobs are released, the jobs go through an additional step before reaching the ring making station. After being released, all the paperwork and raw materials for the job are found and must travel to the ring station for the making of the rings. The process takes 12 minutes on an average according to an exponential distribution for each pair of rings. There are always sufficient workers available to move the rings to the ring station at the beginning of the day. LOTR Makers Inc. would like an analysis of the time to complete the orders for each of the following systems:

- Configuration 1: The system with ring preparation/travel delay with no sharing of the rework craftsman.
- Configuration 2: The system with ring preparation/travel delay and the sharing of the rework craftsman.

Figure 7.64 illustrates the two-system configurations in the form of activity diagrams. Configuration 2 illustrates that the two resources (master craftsman and rework craftsman) are shared at the make ring activity by placing the two resources in a larger oval. This oval represents the fact that these two resources are in a set. Notice how the SEIZE and RELEASE arrows from the make ring activity go to the boundary of the oval. This indicates that the make ring activity pulls resources from this set of resources.

The rework activity still uses the rework craftsman. In particular, the SEIZE and RELEASE arrows go directly to the rework craftsman. The SEIZE arrows have been augmented to indicate that the master craftsman is to be preferred and that the rework



**Figure 7.64** The two LOTR Makers Inc. system configurations. (a) Configuration 1 and (b) configuration 2.

activity has higher priority for the rework craftsman. In both activity diagrams, the preparation/travel time has been represented with an activity that does not use any resources. This can be modeled with a DELAY module. The implementation of configuration 1 poses no additional modeling challenges; however, for configuration 2, the resource sharing must be modeled.

**7.5.1.4 Resource Sets** The diagram indicates that the master craftsman and the rework craftsman are within the same set. Arena™ has the capability of defining sets to hold various object types (e.g., resources and queues). Thus, the set construct can be used to model the sharing of the resources. The set construct is simply a named *list* of objects of the same type. Thus, a resource set is a list of resources. Each resource listed in the set is called a member of the set. Unlike the mathematical set concept, these sets are ordered lists. Each member of the set is associated with an index representing its order in the list. The first member has an index of 1, the second has an index of 2, and so forth. If you know the index, you can look up which member of the set is associated with the index. In this sense, a set is like an array

of objects found in other programming languages. The following illustrates the resource set concept to hold the ring makers.

Index	Member
1	RingMakerR
2	ReworkR

A set named *RingMakers* can be defined with the two previously defined resources as members of the set. In this case, the resource representing the master craftsman (RingMakerR) is placed first in the set and the resource representing the rework craftsman (ReworkR) is placed second in the set. The name of the set can be used to return the associated object:

- RingMakers(1) will return the resource RingMakerR
- RingMakers(2) will return the resource ReworkR

There are three useful functions for working with sets:

**MEMBER(Set ID, Index)** The MEMBER function returns the construct number of a particular set member. Set ID identifies the set to be examined and Index is the index into the set. Using the name of the set with the index number is functionally equivalent to the MEMBER function.

**MEMIDX(Set ID, Member ID)** The MEMIDX function returns the index value of a construct within the specified Set ID. Member ID is the name of the construct.

**NUMMEM(Set ID)** The NUMMEM function returns the number of members in the specified Set ID.

The ordering of the members in the set may be important to the modeling because of how the rules for selecting members from the set are defined. When an entity attempts to seize a member of the resource set, a resource selection rule may be invoked. The rule will be invoked if there is more than one resource idle at the time that the entity attempts to seize a member of the resource set. There are seven default rules and two ways to specify user-defined rules:

**CYC** Selects the first available resource beginning with the successor of the last resource selected. This has the effect of cycling through the resources. For example, if there are 5 members in the set 1,2,3,4,5 and 3 was the last resource selected then 4 will be the next resource selected if it is available.

**POR** Selects the first resource for which the required resource units are available. Each member of the set is checked in the order listed. The order of the list specifies the preferred order of selection.

**LNB** Selects the resource that has the largest number of resource units busy, any ties are broken using the POR rule.

**LRC** Selects the resource that has the largest remaining resource capacity, any ties are broken using the POR rule.

**SNB** Selects the resource that has the smallest number of resource units busy, any ties are broken using the POR rule.

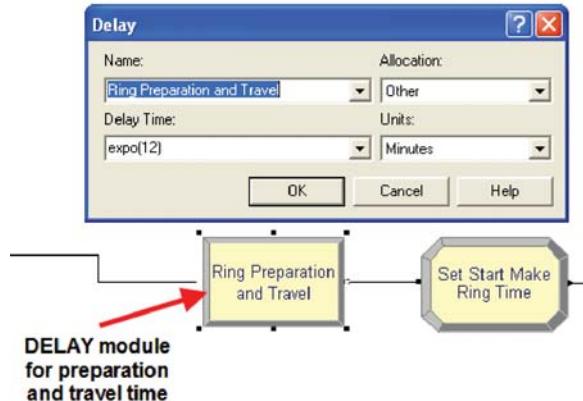


Figure 7.65 DELAY module for preparation/travel time.

**SRC** Selects the resource that has the smallest remaining resource capacity, any ties are broken using the POR rule.

**RAN** Selects randomly from the resources of the set for which the required resource units are available.

**ER(User Rule)** Selects the resource based on the rule User Rule defined in the experiment frame.

**UR(User Rule)** Selects the UR<sub>th</sub> resource where UR is computed in a user-coded rule function.

Since the master ring maker should be preferred if both are available, the RingMakerR resource should be listed first in the set and the POR resource selection rule should be used. The only other modeling issue that must be handled is the fact that the rework worker should show priority for rework jobs.

From the activity diagram, you can see that there will be two SEIZE modules attempting to grab the rework worker. If the rework worker becomes idle, which SEIZE should have preference? According to the problem, the SEIZE related to the rework activity should be given preference. In the module, you can specify a priority level associated with the SEIZE to handle this case. A lower number results in the SEIZE having a higher priority.

Now, you are ready to implement this situation by modifying the file, *LOTRCh7Example.doe*. Open up the submodel named *Ring Processing*, and insert a DELAY module at the beginning of the process as shown in Figure 7.65. The DELAY module can be found on the Advanced Process panel. Specify an expo(12) distribution for the delay time. After making this change, you should save the model under the name *LOTRCh7Config1.doe* to represent the first system configuration. You will now edit this model to create the second system configuration.

The first step will be to define the resource set. This can be done using the SET module on the Basic Process panel. Within the SET module, double-click on a row to start a new set. Then, you should name the set RingMakers and indicate that the type of set is Resource as shown in Figure 7.66. Now, you can click on the Members area to add rows for each member. Do this as shown in Figure 7.66 and make sure to place RingMakerR and ReworkR as the first and second members in the set. Since the resources had already been defined,

it is a simple matter of using the drop-down text box to select the proper resources. If you define a set of resources before defining the resources using the RESOURCE module, the programming environment will automatically create the listed resources in the RESOURCE module. You will still have to edit the RESOURCE module.

After defining and adding the resources to the set, you should save your model as *LOTRCh4Config2.doe*. You are now ready to specify how to use the sets within model. Open up the PROCESS module named Make Ring Process, in order to edit the previously defined resource specification for the SEIZE, DELAY, RELEASE logic. In Figure 7.67, the resource type has been specified as Set. Then, you should select the RingMakers set using the Preferred Order resource selection rule. You should close up the PROCESS module and save your model.

Now, you must handle the fact that the processing time to make a ring depends on which resource is selected. In Figure 7.67, there is a text field labeled Save Attribute. This attribute will hold the index number of the resource that was selected by the SEIZE. In the current situation, this attribute can be used to have the rings (entity) remember which resource was selected. This index will have the value 1 or 2 according to whichever member of the RingMakers set was selected. This index can then be used to determine the appropriate processing time. Since there are only two ring makers, an arrayed EXPRESSION can be defined, see Figure 7.68, to represent the different processing times. The first expression represents the processing time for the master ring maker, and the second expression represents the ring making time of the rework worker. The Save Attribute index can be used to select the appropriate processing time distribution within this arrayed expression. After defining your expressions as shown in Figure 7.68, open up your Make Ring Process module and edit it according to Figure 7.69. Notice how an attribute has been used to remember the index and then that attribute is used to select the appropriate processing time from the EXPRESSION array.

The next required model change is to ensure that the rework craftsman gives priority to rework jobs. This can be done by editing the PROCESS module for the rework process as shown in Figure 7.70. In addition, you need to add an additional 15 minutes for the rework worker's time to perform the rework due to the job sharing. You are now almost ready to run the models.

The final change to the model will enable the time to produce the rings to be captured to a file for analysis within the Output Analyzer (Figure 7.71). Go to the STATISTICS module and add a file name (*prodTimeC2.dat*) to the OUTPUT statistic for the time to make the



**Figure 7.66** Adding members to set.

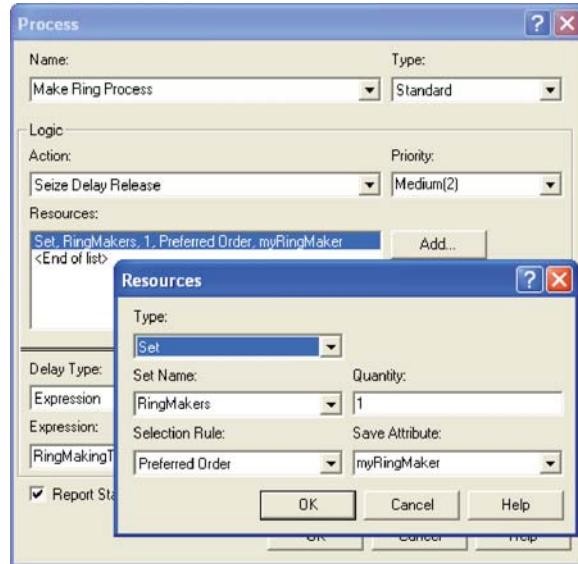


Figure 7.67 Using a resource set in a PROCESS module.

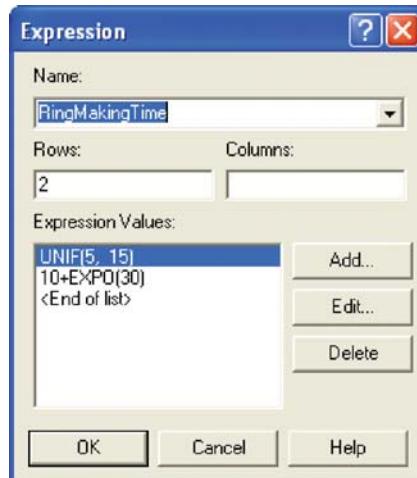


Figure 7.68 Expressions for ring making time by type of worker.

order statistics. The time to make the orders will be written to the file so that the analysis tools within the Output Analyzer can be used. You should then open up the model file for the first configuration and add an output file (e.g., *prodTimeC1.dat*) to capture the statistics for the first configuration. You should then run each model for 30 replications.

After running the models, open up the Output Analyzer and add the two generated files to a new data group. Use the Analyze > Compare Means option of the Output Analyzer to develop a paired confidence interval for the case of common random numbers. Figure 7.72 illustrates how to set up the Compare Means options. Note that configuration 1 is associated with data file A and configuration 2 is associated with data file B. The default behavior

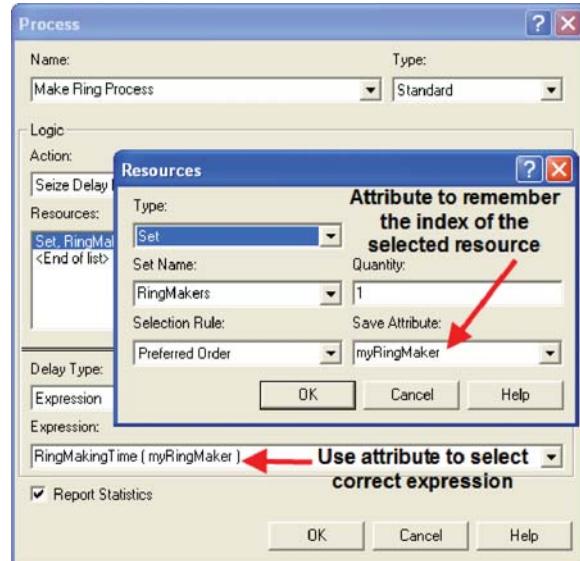


Figure 7.69 Handling processing time by resource selected.

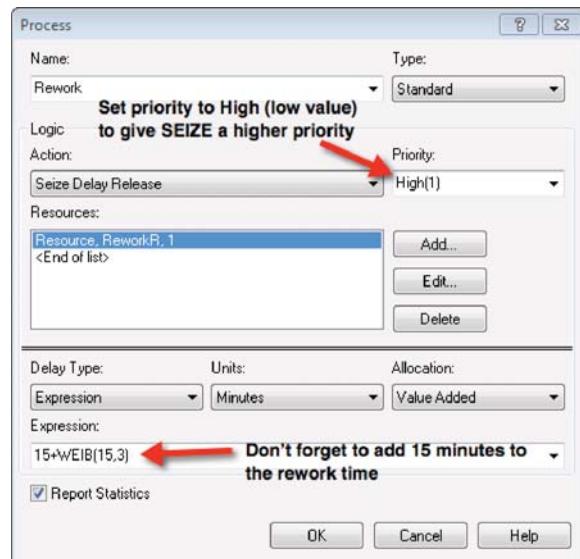
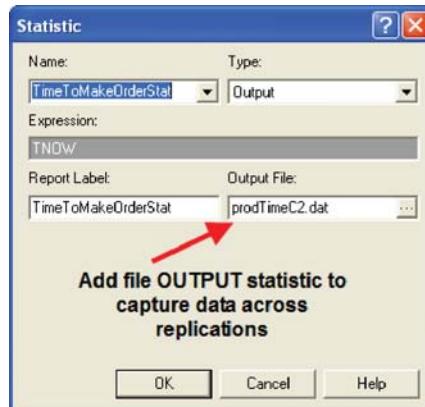


Figure 7.70 Adjusting the priority of a SEIZE.

is to compute the difference between A and B ( $\theta = \theta_1 - \theta_2$ ). Thus, if  $l > 0$ , you can conclude that configuration 1 has the higher mean time to produce the rings. If this is the case, then configuration 2 would be preferred (shorter production time is better). From the results shown in Figure 7.73, you can clearly see that system configuration 2 has the smaller production time. In fact, you can be 95% confident that the true difference between the systems is 92 minutes. This is a practical difference by most standards.



**Figure 7.71** Capturing across replication results.

Based on these results, LOTR Makers, Inc. should consider sharing the rework worker between the work stations. If you check the other performance measures, you will see that the utilization of the rework worker is increased significantly (near 97%) in configuration 2. There is also a larger waiting line at the rework station. Such a high utilization for both the ring makers (especially the rework worker) is a bit worrisome. For example, suppose the quality of the work suffered in the new configuration. Then, there would be even more work for the rework worker and possibly a bottleneck at the rework station. Certainly, such a high utilization is troublesome from a human factors standpoint, since worker breaks have not even been modeled! These and other trade-offs can be examined using simulation (Figures 7.74 and 7.75).

In order to try to ensure a stronger variance reduction when using common random numbers, there are a number of additional implementation techniques that can be applied. For example, to help to ensure that the same random numbers are used for the same processes within each of the simulation alternatives, you should dedicate a different stream number to each random process in the model. To do this, use a different stream number in each probability distribution used within the model. In addition, to help with the synchronization of the use of the random numbers, you can generate the random numbers that each entity will need upon entering the model. Then each entity carries its own random numbers and uses them as it proceeds through the model. In the example, neither technique was done in order to simplify the exposition.

**7.5.1.5 Implementing Independent Sampling** This section outlines how to perform the independent sampling case within a model. The completed model files are available as *LOTRCh7Config1IND.doe* and *LOTRCh7Config2IND.doe*. The basic approach to implement independent sampling is to utilize different random number streams for each configuration. There are a number of different ways to implement the models so that independent sampling can be achieved. The simplest method is to use a variable to represent the stream number in each of the distributions within the model. In *LOTRCh7Config1IND.doe*, every distribution was changed as follows:

- Sales probability distribution:  $BETA(5,1.5, vStream)$
- Success of sale distribution:  $DISC(vSalesProb,1,1.0,0, vStream)$

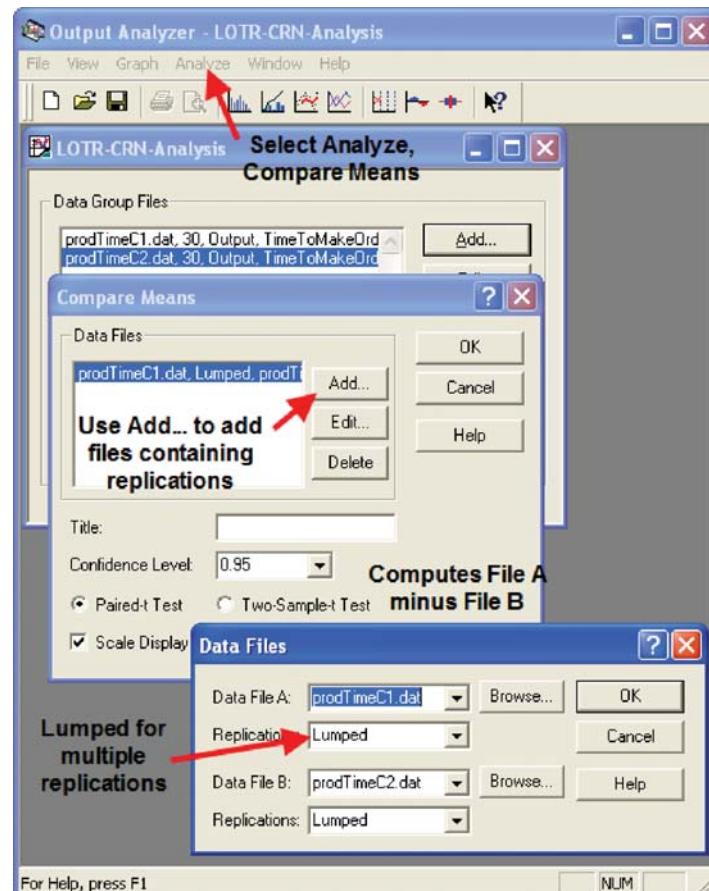


Figure 7.72 Setting up paired difference analysis.

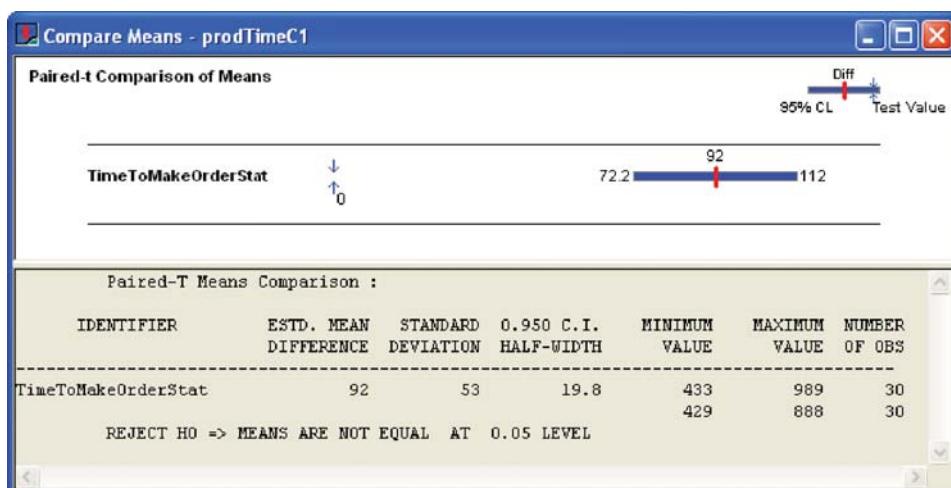


Figure 7.73 Results for comparison of production times.

Instantaneous Utilization	Average	Half Width
InspectorR	0.4282	0.01
PackageR	0.6883	0.01
ReworkR	0.1106	0.02
RingMakerR	0.9785	0.00

**Figure 7.74** Configuration 1 resource utilization.

Instantaneous Utilization	Average	Half Width
InspectorR	0.4832	0.01
PackageR	0.7780	0.02
ReworkR	0.9795	0.00
RingMakerR	0.9221	0.02

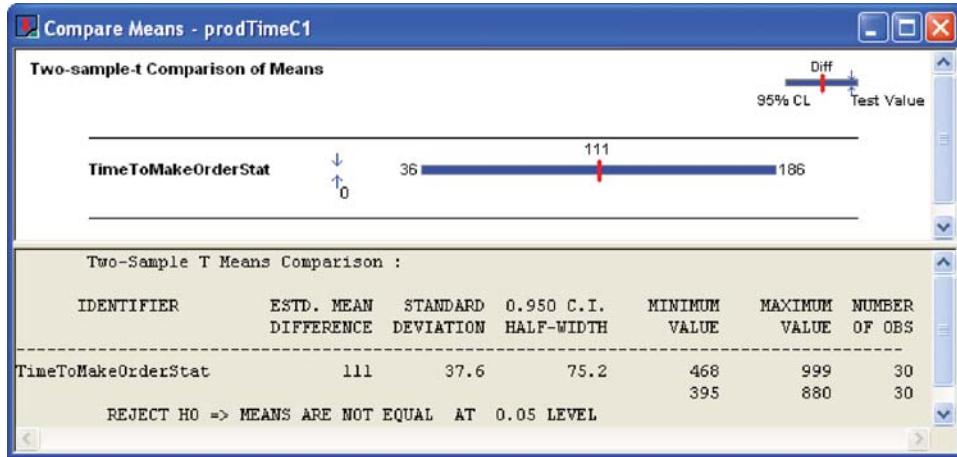
**Figure 7.75** Configuration 2 resource utilization.

- Preparation and travel time distribution: EXPO(12,vStream)
- Master ring make processing time: UNIF(5,15,vStream)
- Inner ring diameter: NORM(vIDM, vIDS, vStream)
- Outer ring diameter: NORM(vODM, vODS, vStream)
- Inspection time distribution: TRIA(2,4,7,vStream)
- Rework time distribution: 5+WEIB(15,3, vStream)
- Packaging time distribution: LOGN(7,1, vStream).

The same procedure was used for *LOTRCh4Config2IND.doe*. Then, the variable vStream was set to different stream numbers so that each model uses different random number streams. Both models were executed first using vStream equal to 1 for configuration 1 and vStream equal to 2 for configuration 2. The Output Analyzer can again be used to compare the results using the Analyze > Compare Means > Two sample *t*-test option. Figure 7.76 presents the results from the analysis.

The results indicate that configuration 2 has a smaller production time. Notice that the confidence interval for the independent analysis is wider than that in the case of common random numbers.

Since comparing two systems through independent samples takes additional work in preparing the model, one may wonder when it should be applied. If the results from one of the alternatives are already available, but the individual replication values are unavailable to perform the pairing, then the independent sample approach might be used. For example, this might be the case if you were comparing to already published results. In addition, suppose the model takes a very long time to execute and only the summary statistics are available for one of the system configurations. You might want to ensure that the running of the second alternative is independent of the first so that you do not have to reexecute the first alternative. Finally, even though the situation of comparing two simulation runs has been the primary focus of this section, you might have the situation of comparing the results of



**Figure 7.76** Independent sample analysis from Output Analyzer.

the simulation to the performance of the actual system. In this case, you can use the data gathered on the actual system and use the two sample-independent analyses to compare the results. This is useful in the context of validating the results of the model against the actual system.

This section presented the results for how to compare two alternatives; however, in many situations, you might have many more than two alternatives to analyze. For a small number of alternatives, you might take the approach of making all pairwise comparisons to develop an ordering. This approach has its limitations, which will be discussed in the next section along with how to handle the multiple comparison of alternative.

### 7.5.2 Analyzing Multiple Systems

The analysis of multiple systems stems from a number of objectives. First, you may want to perform a *sensitivity analysis* on the simulation model. In a sensitivity analysis, you want to measure the effect of small changes to key input parameters to the simulation. For example, you might have assumed a particular arrival rate of customers to the system and want to examine what would happen if the arrival rate decreased or increased by 10%. Sensitivity analysis can help you to understand how robust the model is to variations in assumptions. When you perform a sensitivity analysis, there may be a number of factors that need to be examined in combination with other factors. This may result in a large number of experiments to run. This section discusses how to use the Process Analyzer to analyze multiple experiments. Besides performing a sensitivity analysis, you may want to compare the performance of multiple alternatives in order to choose the best alternative. This type of analysis is performed using multiple comparison statistical techniques. This section also illustrates how to perform a multiple comparison with the best analysis using the Process Analyzer.

**7.5.2.1 Sensitivity Analysis Using the Process Analyzer** For illustrative purposes, a sensitivity analysis on the LOTR Makers, Inc. system will be performed. Suppose that you were concerned about the effect of various factors on the operation of the newly proposed system and that you wanted to understand what happens if these factors vary from the assumed

**TABLE 7.3 Sensitivity Analysis Factors/Levels**

Factor	Description	Base Level	Percentage of Change, %	Low Level	High Level
A	Number of sales calls per day	100	$\pm 10$	90	110
B	Standard deviation of outer ring diameter	0.005	$\pm 2$	0.0049	0.0051
C	Standard deviation of inner ring diameter	0.002	$\pm 2$	0.00196	0.00204

values. In particular, the effects of the following factors in combination with each other are of interest:

- Number of sales calls made each day: What if the number of calls made each day was 10% higher or lower than its current value?
- Standard deviation of inner ring diameter: What if the standard deviation was reduced or increased by 2% compared to its current value?
- Standard deviation of outer ring diameter: What if the standard deviation was reduced or increased by 2% compared to its current value?

The resulting factors and their levels are given in Table 7.3.

The Process Analyzer allows the setting up and the batch running of multiple experiments. With the Process Analyzer, you can control certain input parameters (variables, resource capacities, replication parameters) and define specific response variables (COUNTERS, DSTATS (time-persistent statistics), TALLY (tally-based statistics), OUTPUT statistics) for a given simulation model. An important aspect of using the Process Analyzer is to plan the development of the model so that you can have access to the items that you want to control.

In the current example, the factors that need to vary have already been specified as variables; however, the Process Analyzer has a four decimal place limit on control values. Thus, for specifying the standard deviations for the rings, a little creativity is required. Since the actual levels are simply a percentage of the base level, you can specify the percentage of change as the level of the factor and multiply accordingly in the model. For example, for factor (C) the standard deviation of the inner ring, you can specify 0.98 as the low value since  $0.98 \times 0.002 = 0.00196$ .

Table 7.4 shows the resulting experimental scenarios in terms of the multiplying factor. Within the model, you need to ensure that the variables are multiplied. For example, define three new variables vNCF, vIDF, and vODF to represent the multiplying factors and multiply the original variables where ever they are used:

- In Done Calling? DECIDE module: vNumCalls\*vNCF
- In Determine Diameters ASSIGN module: NORM(vIDM, vIDS\*vIDF, vStream)
- In Determine Diameters ASSIGN module: NORM(vODM, vODS\*vODF, vStream)

Another approach to achieve this would be to define an EXPRESSION and use the expression within the model. For example, you can define an expression *eNumCalls* =

**TABLE 7.4 Combinations of Factors and Levels**

Scenario	vNCF (A)	vIDF (B)	vODF (C)
1	0.90	0.98	0.98
2	0.90	0.98	1.02
3	0.90	1.02	0.98
4	0.90	1.02	1.02
5	1.1	0.98	0.98
6	1.1	0.98	1.02
7	1.1	1.02	0.98
8	1.1	1.02	1.02

$vNumCalls * vNCF$  and use  $eNumCalls$  within the model. The advantage of this is that you do not have to hunt throughout the model for all the changes. The definitions can be easily located within the EXPRESSION module.

The Process Analyzer is a separate program that is installed when the Arena™ environment is installed. Since it is a separate program, it has its own help system, which you can access after starting the program. You can access the Process Analyzer through your Start Menu or through the Tools menu within the Arena™ environment. The first thing to do after starting the Process Analyzer is to start a new PAN file via the File > New menu. You should then see a screen similar to Figure 7.77. In the Process Analyzer, you define the scenarios that you want to be executed. A scenario consists of a model in the form of a (.p) file, a set of controls, and a set of responses. To generate a (.p) file, you can use Run > Check Model. If the check is successful, this will create a (.p) file with the same name as your model to be located in the same directory as the model. Each scenario refers to one (.p) file but the set of scenarios can consist of scenarios that have different (.p) files. Thus, you can easily specify models with different structure as part of the set of scenarios.

To start a scenario, double-click on the add new scenario row within the scenario properties area. You can type in the name for the scenario and the name of the (.p) file (or use the file browser) to specify the scenario as shown in Figure 7.78. Then you can use the Insert menu to insert controls and response variables for the scenario. The insert control dialog is shown in Figure 7.79. Using this, you should define a control for Num Reps (number of replications), vStream (random number stream), vNCF (number of sales calls factor), vIDF (standard deviation of inner ring factor), and vODF (standard deviation of outer ring factor). For each of the controls, be sure to specify the proper data type (e.g., vStream is an Integer and vIDF is a real).

After specifying the first scenario, select the row, right-click, and choose Duplicate Scenario(s) until you have defined eight scenarios. Then, you should complete the scenario definitions as shown in Figure 7.80. To insert a response, use the Insert menu and select the appropriate responses. In this analysis, you will focus on the average time it takes a pair of rings to complete production (*PairOfRings.TotalTime*), the throughput per day (*Throughput*), and the average time spent waiting at the rework station (*Rework.Queue.WaitingTime*).

Since a different stream number has been specified for each of the scenarios, they will all be independent. Within the context of experimental design, this is useful because traditional experimental design techniques (such as response surface analysis) depend on the assumption that the experimental design points are independent. If you use the same stream number for each of the scenarios, then you will be using common random numbers. From the standpoint of the analysis via traditional experimental design techniques, this poses an

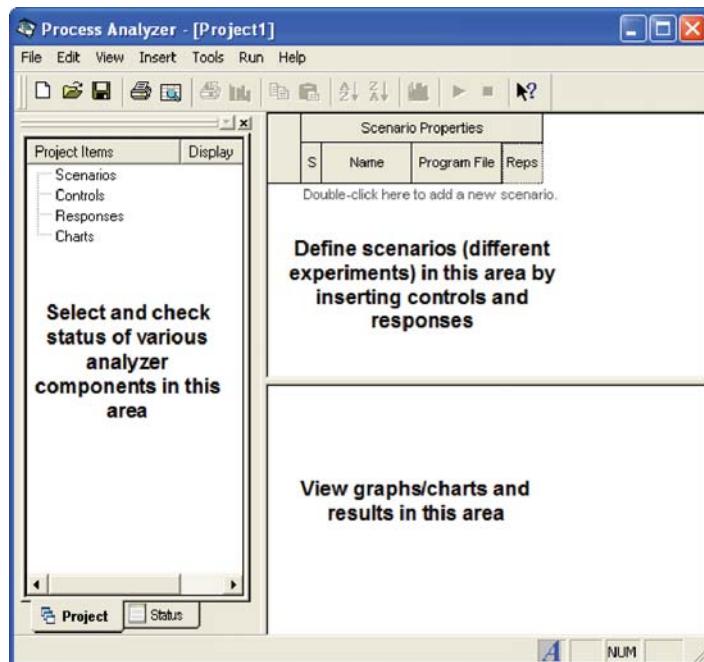


Figure 7.77 The Process Analyzer.



Figure 7.78 Adding a new scenario.

extra complication. The analysis of experimental designs with common random numbers is beyond the scope of this text. The interested reader should refer to Kleijnen [1988] and Kleijnen [1998].

To run all the experiments consecutively, select the scenarios that you want to run and use the Run menu or the VCR like run button. Each scenario will run the number of specified replications and the results for the responses will be tabulated in the response area as shown

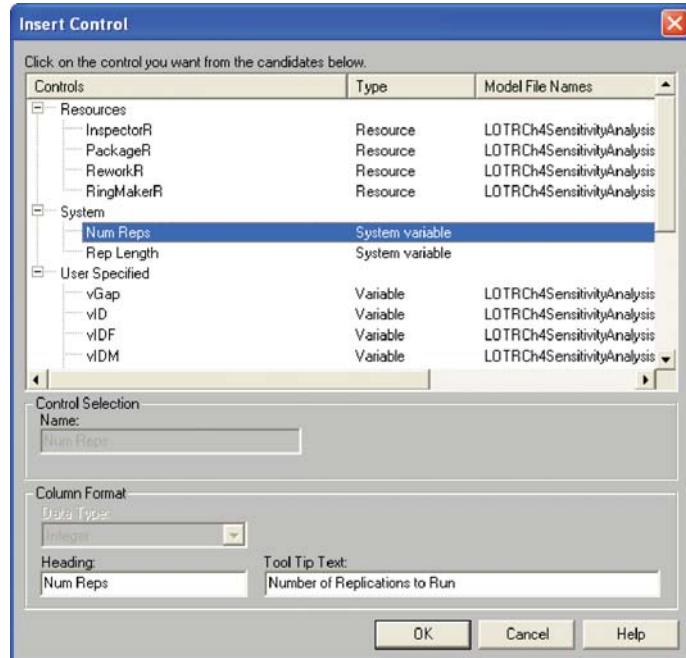


Figure 7.79 Inserting controls.

S	Scenario Properties				Controls					Responses		
	Name	Program File	Reps	Num Reps	vStream	vNCF	vIDF	vODF	Rework.Queu e.WaitingTime	ThroughPut	PairOfRings.T otalTime	
1	Scenario 1	0 : LOTRCh4	0	30	1	0.9000	0.9800	0.9800	---	---	---	
2	Scenario 2	0 : LOTRCh4	0	30	2	0.9000	0.9800	1.0200	---	---	---	
3	Scenario 3	0 : LOTRCh4	0	30	3	0.9000	1.0200	0.9800	---	---	---	
4	Scenario 4	0 : LOTRCh4	0	30	4	0.9000	1.0200	1.0200	---	---	---	
5	Scenario 5	0 : LOTRCh4	0	30	5	1.1000	0.9800	0.9800	---	---	---	
6	Scenario 6	0 : LOTRCh4	0	30	6	1.1000	0.9800	1.0200	---	---	---	
7	Scenario 7	0 : LOTRCh4	0	30	7	1.1000	1.0200	0.9800	---	---	---	
8	Scenario 8	0 : LOTRCh4	0	30	8	1.1000	1.0200	1.0200	---	---	---	

Figure 7.80 Experimental setup controls/responses.

in Figure 7.81. After the simulations have been completed, you can add more responses and the results will be shown.

The purpose in performing a sensitivity analysis is twofold: (i) to see if small changes in the factors result in significant changes in the responses and (ii) to check if the expected direction of change in the response is achieved. Intuitively, if there is low variability in the ring diameters, then you should expect less rework and thus less queuing at the rework station. In addition, if there is more variability in the ring diameters, then you might expect more queuing at the rework station. From the responses for scenarios 1 and 4, this basic intuition is confirmed. The results in Figure 7.81 indicate that scenario 4 has a slightly higher rework waiting time and that scenario 1 has the lowest rework waiting time. Further analysis can be performed to examine the statistical significance of these differences.

S	Scenario Properties			Controls					Responses		
	Name	Program File	Reps	Num Reps	vStream	vNCF	vIDF	vODF	Rework.Queu e.WaitingTime	ThroughPut	PairOfRings.TotalTime
1	Scenario 1	4 : LOTRCh4	30	30	1	0.9000	0.9800	0.9800	27.864	70	314.814
2	Scenario 2	4 : LOTRCh4	30	30	2	0.9000	0.9800	1.0200	29.309	70	314.696
3	Scenario 3	4 : LOTRCh4	30	30	3	0.9000	1.0200	0.9800	36.677	69	311.969
4	Scenario 4	4 : LOTRCh4	30	30	4	0.9000	1.0200	1.0200	36.813	70	322.541
5	Scenario 5	4 : LOTRCh4	30	30	5	1.1000	0.9800	0.9800	35.630	83	370.346
6	Scenario 6	4 : LOTRCh4	30	30	6	1.1000	0.9800	1.0200	33.003	82	366.924
7	Scenario 7	4 : LOTRCh4	30	30	7	1.1000	1.0200	0.9800	31.104	90	400.028
8	Scenario 8	4 : LOTRCh4	30	30	8	1.1000	1.0200	1.0200	36.775	79	359.321

Figure 7.81 Results after running the scenarios.

In general, you should examine the other responses to validate that your simulation is performing as expected for small changes in the levels of the factors. If the simulation does not perform as expected, then you should investigate the reasons. A more formal analysis involving experimental design and analysis techniques may be warranted to ensure statistical confidence in your analysis.

Again, within a simulation context, you know that there should be differences in the responses, so that standard analysis of variance (ANOVA) tests of differences in means are not really meaningful. In other words, they simply tell you whether you took enough samples to detect a difference in the means. Instead, you should be looking at the magnitude and direction of the responses. A full discussion of the techniques of experimental design is beyond the scope of this chapter. For a more detailed presentation of the use of experimental design in simulation, you are referred to Law [2007] and Kleijnen [1998].

Selecting a particular response cell will cause additional statistical information concerning the response to appear in the status bar at the left-hand bottom of the Process Analyzer main window. In addition, if you place your cursor in a particular response cell, you can build charts associated with the individual replications associated with that response. Place your cursor in the cell as shown in Figure 7.81 and right-click to insert a chart. The Chart Wizard will start and walk you through the chart building. In this case, you will make a simple bar chart of the rework waiting times for each of the 30 replications of scenario 1. In the chart wizard, select “Compare the replication values of a response for a single scenario” and choose the Column chart type. Proceed through the chart wizard by selecting Next (and then Finish) making sure that the waiting time is your performance measure. You should see a chart similar to that shown in Figure 7.82.

If you right-click on the chart options pop-up menu, you can gain access to the data and properties of the chart (Figure 7.83). From this dialog, you can copy the data associated with the chart. This gives you an easy mechanism for cutting and pasting the data into another application for additional analysis.

To create a chart across the scenarios, select the column associated with the desired response and right-click. Then, select Insert Chart from the pop-up menu. This will bring up the chart wizard with the “Compare the average values of a response across scenarios” option selected. In this example, you will make a box-and-whiskers plot of the waiting times. Follow the wizard through the process until you have created a chart as shown in Figure 7.84.

There are varying definitions of what constitutes a box-and-whiskers plot. In the Process Analyzer, the box-and-whiskers plot shows whiskers (narrow thin lines) to the minimum and

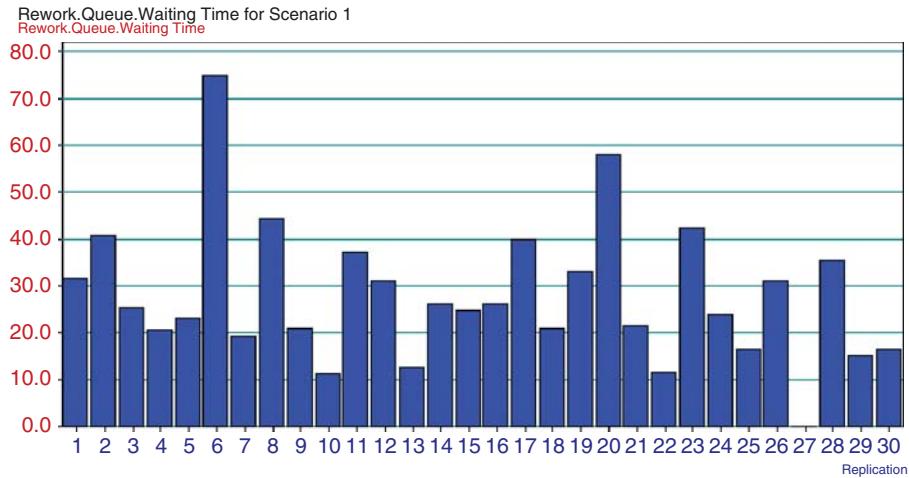


Figure 7.82 Individual scenario chart for rework waiting time.

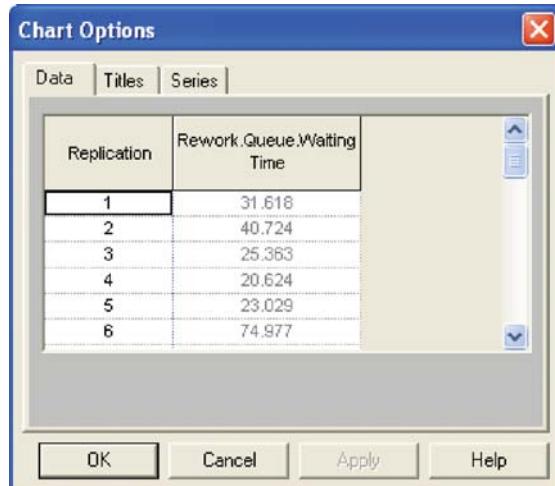


Figure 7.83 Chart options.

maximum of the data and a box (dark solid rectangle) which encapsulates the interquartile range for the data ( third quartile – first quartile). Fifty percent of the data is within the box. This plot can give you an easy way to compare the distribution of the responses across the scenarios. Right-clicking on the chart gives you access to the data used to build the chart, and in particular, it includes the 95% half-widths for confidence intervals on the responses. Table 7.5 shows the data copied from the box-whiskers chart.

This section illustrated how to set up experiments to test the sensitivity of various factors within your models by using the Process Analyzer. After you are satisfied that your simulation model is working as expected, you often want to use the model to help to pick the best design out of a set of alternatives. The case of two alternatives has already been discussed; however, when there are more than two alternatives, more sophisticated statistical techniques are required in order to ensure that a certain level of confidence in the

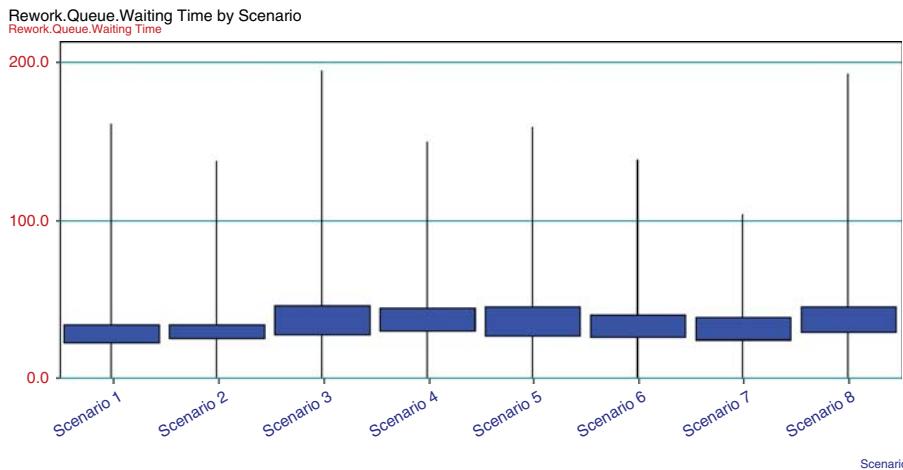


Figure 7.84 Box–whiskers chart across scenarios.

TABLE 7.5 Data from Box–Whiskers Chart

Scenario	Min (A)	Max	Low	High	95% CI
Scenario 1	0	160.8	22.31	33.42	5.53
Scenario 2	0	137.2	24.93	33.69	4.382
Scenario 3	0	194.1	27.47	45.88	9.205
Scenario 4	0	149.8	29.69	43.93	7.12
Scenario 5	0	159.2	26.58	44.68	9.052
Scenario 6	0	138.3	26.11	39.9	6.895
Scenario 7	0	103.5	24	38.2	7.101
Scenario 8	0	193.1	28.82	44.73	7.956

decision-making process is maintained. The next section overviews why more sophisticated techniques are needed. In addition, the section illustrates how to facilitate the analysis using the Process Analyzer.

**7.5.2.2 Multiple Comparisons with the Best** Suppose that you are interested in analyzing  $k$  systems based on performance measures  $\theta_i$ ,  $i = 1, 2, \dots, k$ . The goals may be to compare each of the  $k$  systems with a base case (or existing system), to develop an ordering among the systems, or to select the best system. In any case, assume that some decision will be made based on the analysis and that the risk associated with making a bad decision needs to be controlled. In order to perform an analysis, each  $\theta_i$  must be estimated, which results in sampling error for each individual estimate of  $\theta_i$ . The decision will be based upon all the estimates of  $\theta_i$  (for every system). The sampling error involves the sampling for each configuration. This compounds the risk associated with an overall decision. To see this more formally, the “curse” of the Bonferroni inequality needs to be understood.

The Bonferroni inequality states that given a set of (not necessarily independent) events,  $E_i$ , which occur with probability,  $1 - \alpha_i$ , for  $i = 1, 2, \dots, k$ , then a lower bound on the probability of the intersection of the events is given by

$$P\{\cap_{i=1}^k E_i\} \geq 1 - \sum_{i=1}^k \alpha_i \quad (7.44)$$

In words, the Bonferroni inequality states that the probability of all the events occurring is at least one minus the sum of the probability of the individual events occurring. This inequality can be applied to confidence intervals, which are probability statements concerning the chance that the true parameter falls within intervals formed by the procedure.

Suppose that you have  $c$  confidence intervals each with confidence  $1 - \alpha_i$ . The  $i$ th confidence interval is a statement  $S_i$  that the confidence interval procedure will result in an interval that contains the parameter being estimated. The confidence interval procedure forms intervals such that  $S_i$  will be true with probability  $1 - \alpha_i$ . If you define events,  $E_i = \{S_i \text{ is true}\}$ , then the intersection of the events can be interpreted as the event representing all the statements being true.

$$P\{\text{all } S_i \text{ true}\} = P\{\cap_{i=1}^k E_i\} \geq 1 - \sum_{i=1}^k \alpha_i = 1 - \alpha_E \quad (7.45)$$

where  $\alpha_E = \sum_{i=1}^k \alpha_i$ . The value  $\alpha_E$  is called the overall error probability. This statement can be restated in terms of its complement event as

$$P\{\text{one or more } S_i \text{ are false}\} \leq \alpha_E \quad (7.46)$$

This gives an upper bound on the probability of a false conclusion based on the confidence intervals.

This inequality can be applied to the use of confidence intervals when comparing multiple systems. For example, suppose that you have,  $c = 10$ , 90% confidence intervals to interpret. Thus,  $\alpha_i = 0.10$ , so that

$$\alpha_E = \sum_{i=1}^{10} \alpha_i = \sum_{i=1}^{10} (0.1) = 1.0 \quad (7.47)$$

Thus,  $P\{\text{all } S_i \text{ true}\} \geq 0$  or  $P\{\text{one or more } S_i \text{ are false}\} \leq 1$ . In words, this is implying that the chance that all the confidence intervals procedures result in confidence intervals that cover the true parameter is greater than 0 and less than 1. Think of it this way: If your boss asked you how confident you were in your decision, you would have to say that your confidence is somewhere between 0 and 1. This would not be very reassuring to your boss (or for your job!).

To combat the “curse” of Bonferroni, you can adjust your confidence levels in the individual confidence statements in order to obtain a desired overall risk. For example, suppose that you wanted an overall confidence of 95% on making a correct decision based on the confidence intervals. That is, you desire  $\alpha_E = 0.05$ . You can prespecify the  $\alpha_i$  for each individual confidence interval to whatever values you want, provided that you get an overall

error probability of  $\alpha_E = 0.05$ . The simplest approach is to assume  $\alpha_i = \alpha$ . That is, use a common confidence level for all the confidence intervals. The question then becomes: What should  $\alpha$  be to get  $\alpha_E = 0.05$ ? Assuming that you have  $c$  confidence intervals, this yields

$$\alpha_E = \sum_{i=1}^c \alpha_i = \sum_{i=1}^c \alpha = c\alpha \quad (7.48)$$

So that you should set  $\alpha = \alpha_E/c$ . For the case of  $\alpha_E = 0.05$  and  $c = 10$ , this implies that  $\alpha = 0.005$ . What does this do to the width of the individual confidence intervals? Since the  $\alpha_i = \alpha$  have gotten smaller, the confidence coefficient (e.g.,  $z$  value or  $t$  value) used in confidence interval will be larger, resulting in a wider confidence interval. Thus, you must trade-off your overall decision error against wider (less precise) individual confidence intervals.

Because the Bonferroni inequality does not assume independent events, it can be used for the case of comparing multiple systems when common random numbers are used. In the case of independent sampling for the systems, you can do better than simply bounding the error. For the case of comparing  $k$  systems based on independent sampling, the overall confidence is

$$P\{\text{all } S_i \text{ true}\} = \prod_{i=1}^c (1 - \alpha_i) \quad (7.49)$$

If you are comparing  $k$  systems where one of the systems is the standard (e.g., base case and existing system), you can reduce the number of confidence intervals by analyzing the difference between the other systems and the standard. That is, suppose system 1 is the base case, then you can form confidence intervals on  $\theta_1 - \theta_i$  for  $i = 2, 3, \dots, k$ . Since there are  $k - 1$  differences, there are  $c = k - 1$  confidence intervals to compare.

If you are interested in developing an ordering between all the systems, then one approach is to make all the pairwise comparisons between the systems. That is, construct confidence intervals on  $\theta_j - \theta_i$  for  $i \neq j$ . The number of confidence intervals in this situation is

$$c = \binom{k}{2} = \frac{k(k-1)}{2} \quad (7.50)$$

The trade-off between overall error probability and the width of the individual confidence intervals will become severe in this case for most practical situations.

Because of this problem, a number of techniques have been developed to allow the selection of the best system (or the ranking of the systems) and still guarantee an overall prespecified confidence in the decision. The Process Analyzer uses a method based on multiple comparison procedures as described in Goldsman and Nelson [1998] and the references therein. See also Law [2007] for how these methods relate to other ranking and selection methods.

Using the scenarios that were already defined within the sensitivity analysis section, the following example illustrates how you can select the best system with an overall confidence of 95%. The procedure built into the Process Analyzer can handle common random numbers. The previously described scenarios were set up and reexecuted as shown in Figure 7.85. Notice in the figure that the stream number for each scenario was set to the same value, thereby applying common random numbers. The PAN file for this analysis is called LOTR-MCB.pan and can be found in the supporting files for this chapter.

Suppose you want to pick the best scenario in terms of the average time that a pair of rings spends in the system. Furthermore, suppose that you are indifferent between the systems if they are within 5 minutes of each other. Thus, the goal is to pick the system that has the smallest time with 95% confidences.

To perform this analysis, right-click on the *PairOfRings.TotalTime* response column and choose insert chart. Make sure that you have selected “Compare the average values of a response across scenarios” and select a suitable chart in the first wizard step. This example uses a Hi-Lo chart which displays the confidence intervals for each response and has the minimum and maximum value of each response. The comparison procedure is available with the other charts as well. On the second wizard step, you can pick the response that you want (*PairOfRings.TotalTime*) and choose next. On the third wizard step, you can adjust your titles for your chart. When you get to the fourth wizard step, you have the option of identifying the best scenario. Figure 7.86 illustrates the settings for the current situation. Select the “identify the best scenarios option” with the smaller is better option, and specify an indifference amount (error tolerance) as 5 minutes. Using the Show Best Scenarios button, you can see the best scenarios listed. Clicking Finish causes the chart to be created and the best scenarios to be identified in red (scenarios 1, 2, 3, and 4) as shown in Figure 7.87.

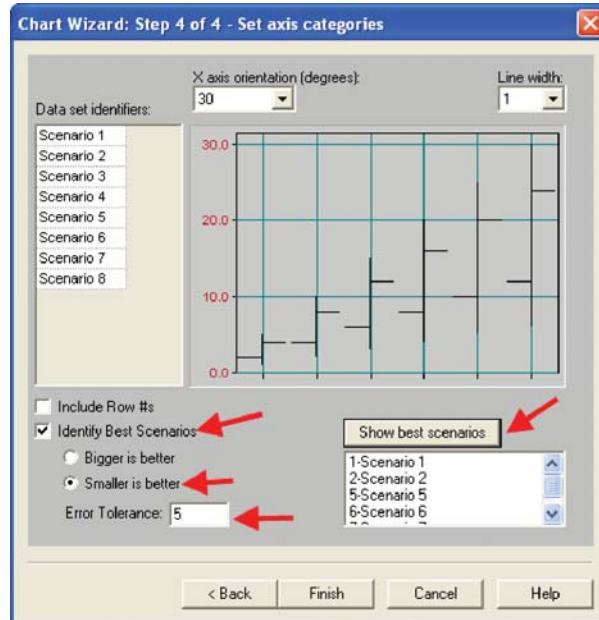
As indicated in Figure 7.87, four possible scenarios have been recommended as the best. This means that you can be 95% confident that any of these four scenarios is the best based on the 5-minute error tolerance. This analysis has narrowed down the set of scenarios, but has not recommended a specific scenario because of the variability and closeness of the responses. To further narrow the set of possible best scenarios down, you can run additional replications of the identified scenarios and adjust your error tolerance. Thus, with the Process Analyzer, you can easily screen systems out of further consideration and adjust your statistical analysis in order to meet the objectives of your simulation study.

## 7.6 SUMMARY

This chapter described many of the statistical aspects of simulation that you will typically encounter in performing a simulation study. An important aspect of performing a correct simulation analysis is to understand the type of data associated with your performance measures (time based vs observation based) and how to collect/analyze such data. Then in your modeling, you will be faced with specifying the time horizon of your simulation. Most situations involve finite horizons, which are fortunately easy to analyze via the method of

S	Scenario Properties			Controls					Responses		<i>PairOfRings.TotalTime</i>
	Name	Program File	Reps	Num Reps	vStream	vNCF	vIDF	vCDF	Rework.Queue.WaitingTime	ThroughPut	
1	Scenario 1	4 : LOTRCh4	30	30	1	0.9000	0.9800	0.9800	27.864	70	314.814
2	Scenario 2	4 : LOTRCh4	30	30	2	0.9000	0.9800	1.0200	29.309	70	314.696
3	Scenario 3	4 : LOTRCh4	30	30	3	0.9000	1.0200	0.9800	36.677	69	311.969
4	Scenario 4	4 : LOTRCh4	30	30	4	0.9000	1.0200	1.0200	36.813	70	322.541
5	Scenario 5	4 : LOTRCh4	30	30	5	1.1000	0.9800	0.9800	35.630	83	370.346
6	Scenario 6	4 : LOTRCh4	30	30	6	1.1000	0.9800	1.0200	33.003	82	366.924
7	Scenario 7	4 : LOTRCh4	30	30	7	1.1000	1.0200	0.9800	31.104	90	400.028
8	Scenario 8	4 : LOTRCh4	30	30	8	1.1000	1.0200	1.0200	36.775	79	359.321

Figure 7.85 Results for MCB analysis using CRN.

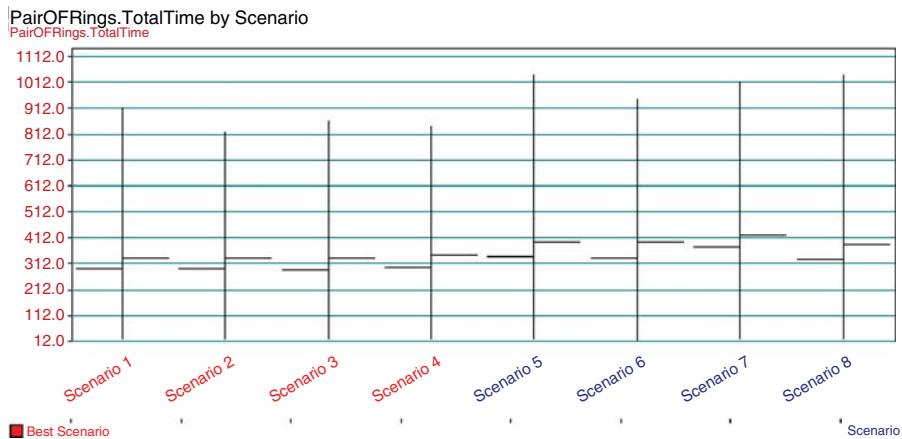


**Figure 7.86** Identifying the best scenario.

replications. This allows a random sample to be formed across replications and to analyze the simulation output via traditional statistical techniques.

In the case of infinite-horizon simulations, things are more complicated. You must first analyze the effect of any warm-up period on the performance measures and decide whether you should use the method of replication–deletion or the method of batch means. Regardless of the situation, Arena™ has tools to handle the situation or facilitates the necessary analysis.

Since you often want to use simulation to make a recommendation concerning a design configuration, an analysis across system configurations must be carefully planned. Arena™



**Figure 7.87** Possible best scenarios.

facilitates the analysis of two or more systems. The Process Analyzer makes the setting up and execution of experimental designs easier and facilitates the ranking and selection of design configurations. When performing your analysis, you should consider how and when to use the method of common random numbers and you should consider the impact of common random numbers on how you analyze the simulation results.

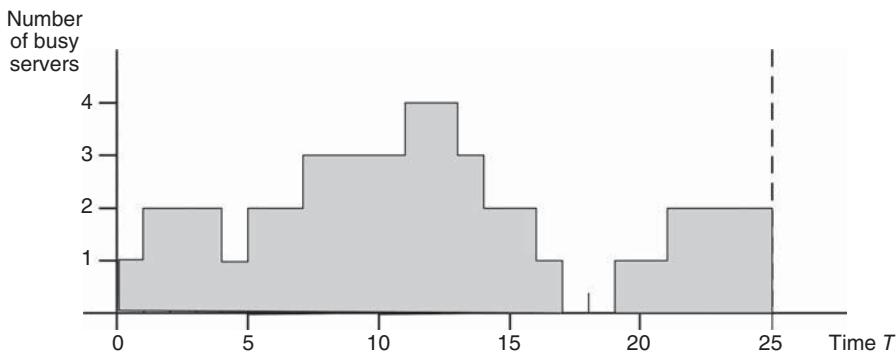
Now that you have a solid understanding of how to program and model in Arena<sup>TM</sup> and how to analyze your results; you are ready to explore the application of Arena<sup>TM</sup> to additional modeling situations involving more complicated systems. The next chapter concentrates on systems involving queuing and inventory components. These systems form the building blocks for modeling more complicated systems in manufacturing, transportation, and service industries.

## EXERCISES

- 7.1 The method of \_\_\_\_\_ assumes that a random sample is formed when repeatedly running the simulation.
- 7.2 The batch means method for analyzing one long simulation run is needed because within-replication data are often (a) \_\_\_\_\_ and (b) \_\_\_\_\_.
- 7.3 The (a)\_\_\_\_\_ conditions of a simulation represent the state of the system when the simulation is started. If we are interested in steady-state performance, then there may be (b)\_\_\_\_\_ in our estimates if we do not account for these conditions. One method to mitigate this problem is to perform a (c)\_\_\_\_\_ analysis to determine an appropriate (d)\_\_\_\_\_ period.
- 7.4 *True or False:* The concept of steady state implies that after a long enough time, the system will not change with respect to time.
- 7.5 *True or False:* To obtain the same overall confidence in a decision involving multiple alternatives, we can reduce the individual confidence levels. This will result in wider individual confidence intervals.
- 7.6 The method of \_\_\_\_\_ assumes that each system configuration is simulated using the same sequences of random numbers.
- 7.7 When performing a \_\_\_\_\_ analysis, we want to measure the effect of small changes to key input parameters to the simulation.
- 7.8 *True or False:* Within a simulation context, we know that there should be differences in the responses for different simulation alternatives. Thus, standard hypothesis tests of equality of means are very appropriate.
- 7.9 Classify each variable as being observational or time persistent:
  - (a) The number of jobs waiting to be processed by a machine.
  - (b) The number of jobs completed during a week.
  - (c) The time that the resource spends serving a customer.
  - (d) The number of items in sitting on a shelf waiting to be sold.
- 7.10 Name two key statistical aspects associated with the waiting times generated from within a replication of a simple queuing situation (e.g., M/M/1) that make standard confidence interval calculation using the student *t*-distribution inappropriate.

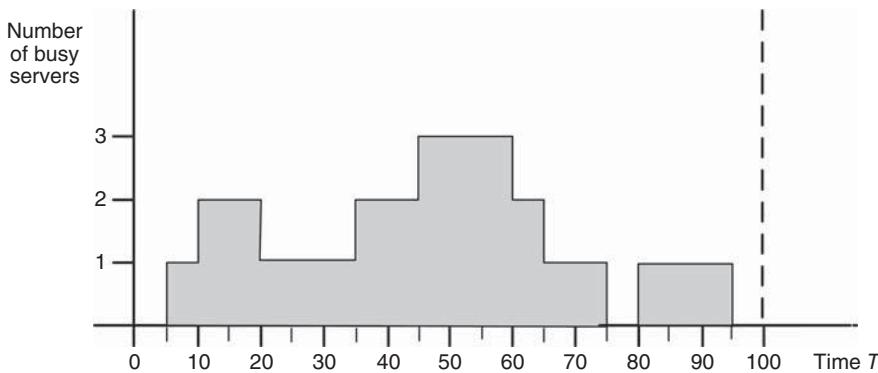
7.11 In a four-server facility, the number of busy servers changes with time as shown below. Compute the following measures of performance over a period of 25 time units.

- Average number of busy servers.
- Average busy time per server.
- Average idle time per server.
- The utilization of the servers.



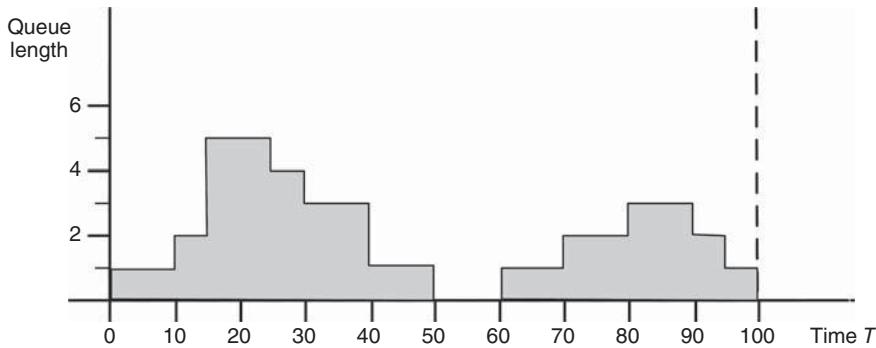
7.12 The usage of a resource with a capacity of 3 can be summarized graphically as shown in the following figure. Compute the following:

- The average utilization.
- The percentage of idleness.
- The average idle time.
- The average busy time.

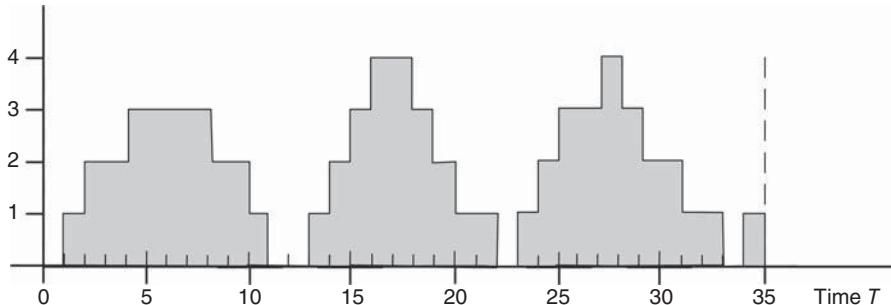


7.13 The following figure summarizes the changes in queue length with time. Compute the following:

- Average queue length.
- Average waiting time in queue for those who must wait.



- 7.14 The following figure shows the changes in queue length for a single-server model over a run length of 35 time units. The first 5 time units are estimated to represent the warm-up period. The remaining 30 time units are divided equally among five batches. The mean and variance of queue length is of interest. For each batch, compute the time-average batch mean of the queue length. Use your results to estimate the mean and variance of queue length.



- 7.15 Using the supplied data set, draw the sample path for the state variable,  $Y(t)$ . Assume that the value of  $Y(t)$  is the value of the state variable just after time  $t$ . Compute the time average over the supplied time range.

$t$	0	1	6	10	15	18	20	25	30	34	39	42
$Y(t)$	1	2	1	1	1	2	2	3	2	1	0	1

- 7.16 Using the supplied data set, draw the sample path for the state variable,  $N(t)$ . Give a formula for estimating the time average number in the system,  $N(t)$ , and then use the data to compute the time average number in the system over the range from 0 to 25. Assume that the value of  $N(t)$  is the value of the state variable just after time  $t$ .

$t$	0	2	4	5	7	10	12	15	20
$N(t)$	0	1	0	1	2	3	2	1	0

- 7.17 Using the supplied data set, draw the sample path for the state variable,  $N(t)$ .

$t$	0	2	4	5	7	10	12	15	20
$N(t)$	0	1	0	1	2	3	2	1	0

- (a) Using a batching interval of 5, apply the batch means method to estimate the average number of customers in the system over the range from 0 to 25.
- (b) Give a formula for estimating the mean rate of arrivals over the interval from 0 to 25 and then use the data to estimate the mean arrival rate.
- (c) Estimate the average time in the system (waiting and in service) for the customers indicated in the diagram.
- 7.18 Compute the required sample size necessary to ensure a 95% confidence interval with a half-width of no larger than 30 minutes for the total time to produce the rings for the LOTR System of Section 7.3.2. Use the following three methods:
- Sample size based on the normal distribution.
  - Sample size based on the *t*-distribution approximation.
  - Sample size using the half-width ratio method. Discuss the differences between the methods. Run the model for the specified number of replications and report your results. Did the half-width from your replications meet the target value? Discuss.
- 7.19 Assume that the following Arena output represents the summary statistics for a pilot run of 10 replications from a simulation for the system time in minutes.

Replications:	10	Time Units :	Minutes
<b>User Specified</b>			
<b>Tally</b>			
Interval	Average	Half Width	
Record System Time	78.2658	9.39	

Find the approximate number of additional replications to execute in order to have a 99% confidence interval that is within  $\pm 2$  minutes of the true mean system time using the half-width ratio method.

- 7.20 Using the Lindley equation spreadsheet simulation, perform the following:
- Develop a 95% confidence interval for your estimate of the mean waiting time based on the data from one replication. Discuss why this is inappropriate. How does your simulation estimate compare to the theoretical value?
  - How does your running average track the theoretical value? What would happen if you increased the number of customers?
  - Construct the Welch plot using five replications of the 1000 customers. Determine a warm-up point for this simulation. Do you think that 1000 customers are enough?
  - Make an autocorrelation plot of your 1000 customer wait times using your favorite statistical analysis package. What are the assumptions for forming the confidence interval in part (a). Is this data independent and identically distributed? What is the implication of your answer for your confidence interval in part (a)?

- (e) Use your warm-up period from part (c) and generate an addition 1000 customers after the warm-up point. Use the method of batch means to batch the 1000 observations into 40 batches of size 25. Make an autocorrelation plot of the 40 batch means. Compute a 95% confidence interval for the mean waiting time using the 40 batches.
- (f) Use the method of replication deletion to develop a 95% confidence interval for the mean waiting time. Use your warm-up period from part (c). Compare the result with that of (a) and (e) and discuss.
- 7.21 Create a spreadsheet simulation to simulate observations from a  $N(\mu, \sigma^2)$  random variable.
- Use your simulation to generate two independent samples of size  $n_1 = 20$  and  $n_2 = 30$  from normal distributions having  $\mu_1 = 2, \sigma_1^2 = 0.64, \mu_2 = 2.2, \sigma_2^2 = 0.64$ . Assume that you do not know the true means and variances. Use the method of independent samples to examine whether  $\mu_2 > \mu_1$ . Show all your work.
  - Use your simulation to generate two independent samples of size  $n_1 = 20$  and  $n_2 = 30$  from normal distributions having  $\mu_1 = 2, \sigma_1^2 = 0.64, \mu_2 = 2.2, \sigma_2^2 = 0.36$ . Assume that you do not know the true means and variances. Use the method of independent samples to examine whether  $\mu_2 > \mu_1$ . Show all your work.
  - Use your simulation to generate two independent samples of size  $n_1 = 30$  and  $n_2 = 30$  from normal distributions having  $\mu_1 = 2, \sigma_1^2 = 0.64, \mu_2 = 2.2, \sigma_2^2 = 0.36$ . Assume that you do not know the true means and variances. Use the paired-*t* method to examine whether  $\mu_2 > \mu_1$ .
  - Repeat part (c) but instead generate two dependent samples. Use the paired-*t* method to examine whether  $\mu_2 > \mu_1$ . What is the effect of using common random numbers? Show all your work.
- 7.22 Consider a manufacturing system comprising two different machines and one operator. The single operator is shared between the two machines. Parts arrive with an exponentially distributed interarrival time with a mean of 3 minutes. The arriving parts are one of two types. Sixty percent of the arriving parts are Type 1 and are processed on Machine 1. These parts require the assigned operator for a 1-minute setup operation. The remaining 40% of the parts are Type 2 parts and are processed on Machine 2. These parts require the assigned operator for a 1.5-minute setup operation. The service times (excluding the setup time) are lognormally distributed with a mean of 4.5 minutes and a standard deviation of 1 minute for Type 1 parts and a mean of 7.5 minutes and a standard deviation of 1.5 minutes for Type 2 parts. Run your model for 20,000 minutes, with 20 replications. Report the utilization of the machines and operators. In addition, report the total time spent in the system for each type of part.
- 7.23 Recall Exercise (5.24). Determine the number of days (replications) to run so that you can be 95% confident that you are within 2 minutes of the true mean time in the system for damaged orders. Rerun the model and summarize your statistics in a table.
- What would happen if the arrival rate were to double (i.e., the interarrival time mean were 5 minutes instead of 10 minutes)? In this case, if you could place another person anywhere in the system to help out with one of the five tasks, where would you place the person? Justify your answer with simulation results.

- (b) Report confidence interval statistics on the number of suits left in the system at the end of the 12-hour shift. In other words, how many suits on an average are left for the skeleton crew to finish up?
- 7.24 YBox video game players arrive at a two-person station for testing. The inspection time per YBox set is EXPO(10) minutes. On an average 82% of the sets pass inspection. The remaining 18% is routed to an adjustment station with a single operator. Adjustment time per YBox is UNIF(7,14) minutes. After adjustments are made, the units are routed back to the inspection station to be retested. Build an Arena™ simulation model of this system. Use a replication length of 30,000 minutes.
- (a) Perform a warm-up analysis of the total time a set spends in the system and estimate the system time to within 2 minutes with 95% confidence.
  - (b) Collect statistics to estimate the average number of times a given job is adjusted.
  - (c) Suppose that any one job is not allowed more than two adjustments, after which time the job must be discarded. Modify your simulation model and estimate the number of discarded jobs.
- 7.25 Cars arrive every EXPO(18) minutes at a car-wash facility that also offers vacuum cleaning. It takes EXPO(12) minutes to wash and EXPO(15) minutes to vacuum clean. When a car arrives, it can go to either wash or vacuum cleaning, depending on which queue is shorter. After the first activity (wash or clean), the car must go to the remaining activity (clean or wash). Assuming infinite queue sizes, determine the average time a car spends in washing and the average time a car spends in cleaning, as well as the time it spends in the system using a simulation model built using Arena™. Include a warm-up analysis of the system time and estimate the system time to within 2 minutes with 95% confidence.
- 7.26 Jobs arrive in batches of 10 items each. The interarrival time is EXPO(2) hours. The machine shop contains two milling machines and one drill press. About 30% of the items require drilling before being processed on the milling machine. Drilling time per item is UNIF(10, 15) minutes. The milling time is EXPO(15) minutes for items that do not require drilling, and UNIF(15,20) for items that do. Assume that the shop has two 8-hour shifts each day and that you are only interested in the *first* shift's performance. Any jobs left over at the end of the first shift are left to be processed by the second shift. Estimate the average number of jobs left for the second shift to complete at the end of the first shift to within plus or minus five jobs with 95% confidence. What is your replication length? Number of replications? Determine the utilization of the drill press and the milling machines as well as the average time an item spends in the system.
- 7.27 A repair and inspection facility consists of two stations, a repair station with two technicians and an inspection station with 1 inspector. Each repair technician works at a rate of three items per hour, while the inspector can inspect eight items per hour each exponentially distributed. Approximately 10% of all items fails inspection and is sent back to the repair station (this percentage holds even for items that have been repaired two to three times). If an item fails inspection three times, then it is scrapped. When an item is scrapped, the item is sent to a disassembly station to recover the usable parts. At the disassembly station, the items wait in a queue until a technician is available. The disassembly time is distributed according to a lognormal distribution with a mean of 20 minutes and a standard deviation of 10 minutes. Assume that items

arrive according to a Poisson arrival process with a rate of 4 per hour. The weekly performance of the system is the key objective of this simulation analysis. Assume that the system starts empty and idle on Monday mornings and runs continuously for two shifts per day for 5 days. Any jobs not completed by the end of second shift are carried over to the first shift of the next day. Any jobs left over at the end of the week are handled by a separate weekend staff that is not of concern to the current study. Estimate the following:

- The average system time of items that pass inspection on the first attempt. Measure this quantity such that you are 95% confident to within  $\pm 3$  minutes.
  - The average number of jobs completed per week.
    - a. Sketch an activity diagram for this situation.
    - b. Assume that there are two technicians at the repair station, one inspector at the inspection station, and one technician at the disassembly station. Develop an Arena<sup>TM</sup> model for this situation.
    - c. Assume that there are two technicians at the repair station and one inspector at the inspection station. The disassembly station is also staffed by the two technicians who are assigned to the repair station. Develop an Arena<sup>TM</sup> simulation model for this situation.
- 7.28 A small manufacturing system produces three types of parts. There is a 30% chance of getting a Type 1 part, a 50% chance of getting a Type 2 part and a 20% chance of getting a Type 3 part. The parts arrive from an upstream process such that the time between arrivals is exponentially distributed with a mean of 3 minutes. All parts that enter the system must go through a preparation station where there are two preparation workers. The preparation time is exponentially distributed with means 3, 5, and 7 for part Types 1, 2, and 3, respectively.

There is only space for six parts in the preparation queue. Any parts that arrive to the system when there are six or more parts in the preparation queue cannot enter the system. These parts are shunted to a recirculating conveyor, which takes 10 minutes to recirculate the parts before they can try again to enter the preparation queue. Hint: Model the recirculating conveyor as a simple deterministic delay.

After preparation, the parts are processed on two different production lines. A production line is dedicated to Type 1 parts and a production line is dedicated to Type 2 and 3 parts. Part Types 2 and 3 are built one at a time on their line by one of four operators assigned to the build station. The time to build a part Type 2 or 3 part is triangularly distributed with a (min = 5, mode = 10, max = 15) minutes. After the parts are built, they leave the system.

Part Type 1 has a more complicated process because of some special tooling that is required during the build process. In addition, the build process is separated into two different operations. Before starting operation 1, the part must have 1 of 10 special tooling fixtures. It takes between 1 and 2 minutes uniformly distributed to place the part in the tooling fixture. An automated machine places the part in the tooling so that the operator at operation 1 does not have to handle the part. There is a single operator at operation 1 which takes 3 minutes on an average exponentially distributed. The part remains in the tooling fixture after the first operation and proceeds to the second operation. There is one operator at the second operation which takes between 3 and 6 minutes uniformly distributed. After the second operation is complete, the part must be removed from the tooling fixture. An automated machine removes the part from

the tooling so that the operator at operation 2 does not have to handle the part. It takes between 20 and 30 seconds uniformly distributed to remove the part from the tooling fixture. After the part is built, it leaves the system.

In this problem, the steady-state performance of this system is required in order to identify potential long-term bottlenecks in this process. For this analysis, collect statistics on the following quantities:

- Queue statistics for all stations. Utilization statistics for all resources.
- The system time of parts by part type. The system time should not include the time spent on the recirculating conveyor.
- The average number of parts on the conveyor.

Perform a warm-up analysis on the system time of a part regardless of type.

- 7.29 Reconsider Exercise (7.28). A process change is being recommended for the build station for part Type 2 and 3. In particular, a machine change will cause the processing time to be lognormally distributed with a mean of 10 and a standard deviation of 2 minutes. Use the Output Analyzer to compare the system time of the old configuration and the new configuration based on 30 replications of length 1000 hours with a warm up of 200 hours. Which configuration would you recommend?
- 7.30 Reconsider Exercise (7.28). Suppose that you are interested in checking the sensitivity of a number of configurations and possibly picking the best configuration based on the part system time. The factors of interest are given in the following table.

Factor	Levels
Preparation space	4 or 8
Number of tooling fixtures	10 or 15
Part 2 and 3 processing distribution	TRIA(5, 10, 15) or LOGN(10, 2)

- (a) Use the Process Analyzer to examine these system configurations within an experimental design.
- (b) Based on system time, recommend a configuration that has the smallest system time with 95% confidence on your overall decision. Base your analysis on 30 replications of length 1000 hours and a warm-up period of 200 hours.
- 7.31 As part of a diabetes prevention program, a clinic is considering setting up a screening service in a local mall. They are considering two designs: Design A: After waiting in a single line, each walk-in patient is served by one of three available nurses. Each nurse has their own booth, where the patient is first asked some medical health questions, then the patient's blood pressure and vitals are taken, and finally, a glucose test is performed to check for diabetes. In this design, each nurse performs the tasks in sequence for the patient. If the glucose test indicates a chance of diabetes, the patient is sent to a separate clerk to schedule a follow-up at the clinic. If the test is not positive, then the patient departs.

Design B: After waiting in a single line, each walk-in is served in order by a clerk who takes the patient's health information, a nurse who takes the patient's blood pressure and vitals, and another nurse who performs the diabetes test. If the glucose test indicates a chance of diabetes, the patient is sent to a separate clerk to schedule a

follow-up at the clinic. If the test is not positive, then the patient departs. In this configuration, there is no room for the patient to wait between the tasks; therefore, a patient who had their health information taken cannot move ahead unless the nurse taking the vital signs is available. Also, a patient having their glucose tested must leave that station before the patient in blood pressure and vital checking can move ahead.

Patients arrive to the system according to a Poisson arrival process at a rate of 9.5 per hour (stream 1). Assume that there is a 5% chance (stream 2) that the glucose test will be positive. For design A, the time that it takes to have the paperwork completed, the vitals taken, and the glucose tested are all lognormally distributed with means of 6.5, 6.0, and 5.5 minutes, respectively (streams 3, 4, 5). They all have a standard deviation of approximately 0.5 minutes. For design B, because of the specialization of the tasks, it is expected that the mean of the task times will decrease by 10%.

Assume that the mall opens at 10 am and that the system operates until 8 pm. During the time from 10 am to 12 noon, the arrivals are less than the overall arrival rate. From 10 am to noon, the rate is only 6.5 per hour. From noon to 2 pm, the rate increases to 12.5 per hour. From 2 pm to 4 pm the traffic lightens up again, back to 6.5 per hour. From 4 pm to 6 pm, the rate is 12.5 per hour and finally from 6 pm to 8 pm, the rate is 9.5 per hour. Assume that the clinic is open from 10 am to 8 pm (10 hours each day) and that any patients in the clinic before 8 pm are still served.

The distribution used to model the time that it takes to schedule a follow-up visit is a WEIB(2.6, 7.3) distribution.

Make a statistically valid recommendation as to the best design based on the average system time of the patients. We want to be 95% confident on our recommendation to within 2 minutes.

---

# 8

---

## MODELING QUEUING AND INVENTORY SYSTEMS

### LEARNING OBJECTIVES

- To be able to define and explain the key performance measures within queuing and inventory systems.
- To be able to identify and apply standard queuing models.
- To be able to simulate variations of standard queuing models.
- To be able to simulate networks of queues using the STATION, ROUTE, and SEQUENCE modules.
- To know and understand the capabilities of OptQuest.
- To be able to use the HOLD and SIGNAL modules.
- To be able to simulate classic inventory models
- To be able to simulate multi-echelon inventory systems.

### 8.1 INTRODUCTION

This chapter examines how to model queuing and inventory systems. As illustrated in previous chapters, many real-life situations involve the possible waiting of entities (e.g., customers and parts) for resources (e.g., bank tellers and machines). Systems that involve waiting lines are called queuing systems. In addition to queuing systems, situations involving inventory will also be studied. In the classic sense, inventory is a build up of items for

future allocation to customers. Queuing systems and inventory systems are closely related because inventory can be considered a resource within a system. Just like a customer arriving to a pharmacy to request service from the pharmacy, customers can arrive to a store to request units of an item. These requests are called demands and if the item is unavailable then the customer might wait until a sufficient quantity of the item becomes available. This chapter introduces both analytical (formula-based) and simulation-based approaches to model the performance of these systems.

Once the performance of the system is modeled, the design of the system to meet operational requirements becomes an important issue. For example, in the simple pharmacy example, you might want to determine the number of waiting spaces that should be available so that arriving customers can have a high chance of entering the line. In the case of inventory systems, you might want to design the system so that the quantity of inventory on hand is sufficient to meet customer demand with a high chance that the customer will be able to get the item requested on the first request. In both these situations, having more of the resource (pharmacist or inventory) available at any time will assist in meeting the design criteria; however, an increase in a resource typically comes at some cost. Thus, design questions within queuing and inventory systems involve a fundamental trade-off between customer service and the cost of providing that service.

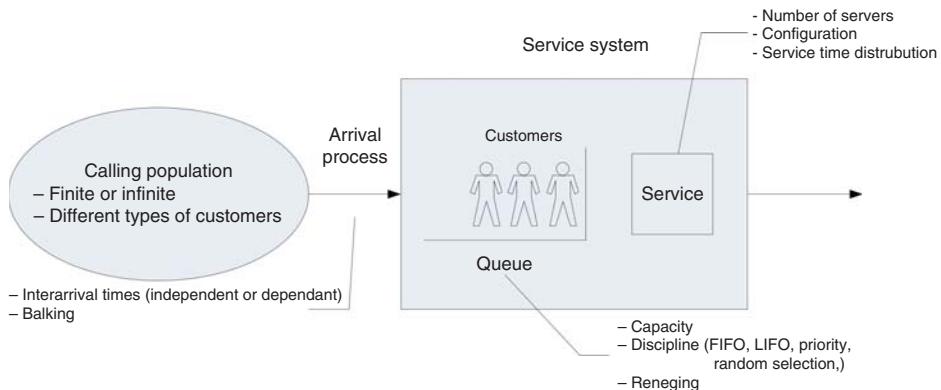
To begin the analysis of these systems, a brief analytical treatment of the key modeling issues is presented. In both cases (queuing and inventory), analytical results are available only for simplified situations; however, the analytical treatment will serve two purposes. First, it will provide an introduction to the key modeling issues, and second, it can provide approximate models for more complicated situations. After understanding some of the basic models, the chapter examines situations for which simulation is the most appropriate mechanism for estimating performance. Along the way, some of the design issues relevant to these systems will be illustrated.

## 8.2 SINGLE LINE QUEUING STATIONS

Chapter 4 presented the pharmacy model and analyzed it with a single-server, single-queue queuing system called the M/M/1. This section shows how the formulas for the M/M/1 model in Chapter 4 were derived and discusses the key notation and assumptions of analytical models for systems with a single queue. In addition, you will also learn how to simulate variations of these models.

In a queuing system, there are customers who compete for resources by moving through processes. The competition for resources causes waiting lines (queues) to form and delays to occur within the customer's process. In these systems, the arrivals and/or service processes are often stochastic. Queuing theory is a branch of mathematical analysis of systems that involve waiting lines in order to predict (and control) their behavior over time. The basic models within queuing theory involve a single line that is served by a set of servers. Figure 8.1 illustrates the major components of a queuing system with a single-queue feeding into a set of servers.

In queuing theory, the term customer is used as a generic term to describe the entities that flow and receive service. A resource is a generic term used to describe the components of the system that are required by a customer as the customer moves through the system. The individual units of the resource are often called servers. In Figure 8.1, the potential customers can be described as coming from a calling population. The term calling population



**Figure 8.1** Single-queue system figure.

comes from the historical use of queuing models in the analysis of phone calls to telephone trunk lines. Customers within the calling population may arrive to the system according to an arrival process. In the finite population case, the arrival rate that the system experiences will quite naturally decrease as customers arrive since fewer customers are available to arrive if they are within the system.

In the infinite calling population case, the rate of arrivals to the system does not depend on how many customers have already arrived. In other words, there are so many potential customers in the population that the arrival rate to the system is not affected by the current number of customers within the system. Besides characterizing the arrival process by the rate of arrivals, it is useful to think in terms of the interarrival times and in particular the interarrival time distribution. In general, the calling population may also have different types of customers who arrive at different rates. In the analytical analysis presented here, there will only be one type of customer.

The queue is that portion of the system that holds waiting customers. The two main characteristics for the queue are its size or capacity and its discipline. If the queue has a finite capacity, this indicates that there is only enough space in the queue for a certain number of customers to be waiting at any given time. The queue discipline refers to the rule that will be used to decide the order of the customers within the queue. A first-come, first-served (FCFS) queue discipline orders the queue by the order of arrival, with the most recent arrival always joining the end of the queue. A last-in, first-out (LIFO) queue discipline has the most recent arrival joining the beginning of the queue. A LIFO queue discipline acts like a stack of dishes. The first dish is at the bottom of the stack, and the last dish added to the stack is at the top of the stack. Thus, when a dish is needed, the next dish to be used is the last one added to the stack. This type of discipline often appears in manufacturing settings when the items are placed in bins, with newly arriving items being placed on top of items that have previously arrived.

Other disciplines include random and priority. You can think of a random discipline modeling the situation of a server randomly picking the next part to work on (e.g., reaches into a shallow bin and picks the next part). A priority discipline allows the customers to be ordered within the queue by a specified priority or characteristic. For example, the waiting items may be arranged by due date for a customer order.

The resource is that portion of the system that holds customers who are receiving service. Each customer who arrives to the system may require a particular number of units of the

resource. The resource component of this system can have one or more servers. In the analytical treatment, each customer will require only one server and the service time will be governed by a probability distribution called the service time distribution. In addition, for the analytical analysis, the service time distribution will be the same for all customers. Thus, the servers are all identical in how they operate and there is no reason to distinguish between them. Queuing systems that contain servers that operate in this manner are often referred to as parallel server systems. When a customer arrives to the system, the customer will be placed either in the queue or in service. After waiting in line, the customer must select one of the (identical) servers to receive service. The following analytical analysis assumes that the only way for the customer to depart the system is to receive service.

### 8.2.1 Queuing Notation

The specification of how the major components of the system operate gives a basic system configuration. To help in classifying and identifying the appropriate modeling situations, Kendall's notation (Kendall [1953]) can be used. The basic format is

arrival process / service process / number of servers / system capacity / size of the calling population / queue discipline

For example, the notation M/M/1 specifies that the arrival process is Markovian (M) (exponential time between arrivals), the service process is Markovian (M) (exponentially distributed service times, and there is one server. When the system capacity is not specified, it is assumed to be infinite. Thus, in this case, there is a single queue that can hold any customer who arrives. The calling population is also assumed to be infinite if not explicitly specified. Unless otherwise noted, the queue discipline is assumed to be FCFS.

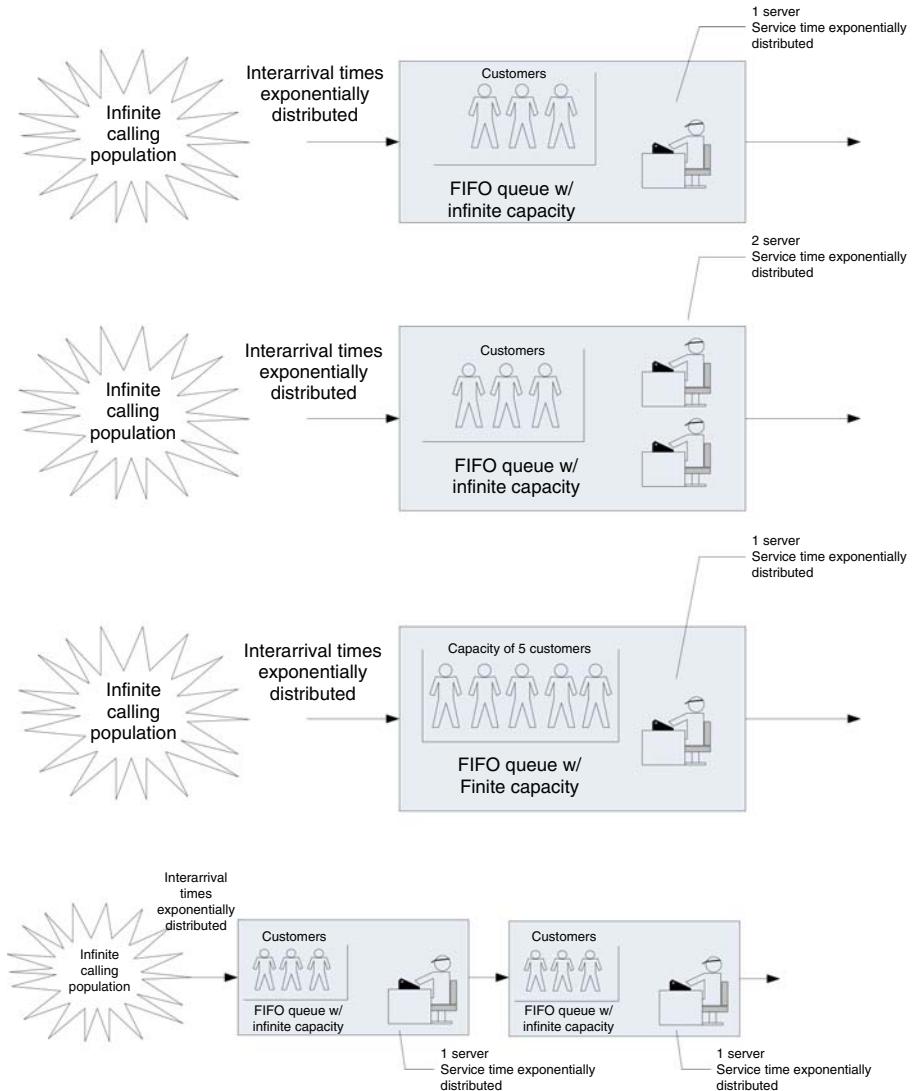
Traditionally, the first letter(s) of the appropriate distribution is used to denote the arrival and service processes. Thus, the case LN/D/2 represents a queue with two servers having a lognormally (LN) distributed time between arrivals and deterministic (D) service times. Unless otherwise specified, it is typically assumed that the arrival process is a renewal process, that is, the time between arrivals are independent and identically distributed (IID) and the service times are also IID.

Also, the standard models assume that the arrival process is independent of the service process and vice versa. To denote any distribution, the letter G for general (any) distribution is used. The notation (GI) is often used to indicate a general distribution in which the random variables are independent. Thus, the G/G/5 represents a queue with an arrival process having any general distribution, a general distribution for the service times, and five servers.

There are a number of quantities that form the basis for measuring the performance of queuing systems.

- The time that a customer spends waiting in the queue:  $T_q$
- The time that a customer spends in the system (queue time plus service time):  $T$
- The number of customers in the queue at time  $t$ :  $N_q(t)$
- The number of customers that are in the system at time  $t$ :  $N(t)$
- The number of customer in service at time  $t$ :  $N_b(t)$ .

These quantities will be random variables when the queuing system has stochastic elements. We will assume that the system is work conserving, that is, the customers do not exit



**Figure 8.2** Illustrations of some common queuing situations.

without received all of their required service and a customer uses one and only one server to receive service. For a work conserving queue, the following is true.

$$N(t) = N_q(t) + N_b(t) \quad (8.1)$$

This indicates that the number of customers in the system must be equal to the number of customers in queue plus the number of customers in service. Since the number of servers is known, the number of busy servers can be determined from the number of customers in the system. Under the assumption that each customer requires 1 server (1 unit of the resource), then  $N_b(t)$  is also the current number of busy servers. For example, if the number of servers is 3 and the number of customers in the system is 2, then there must be two customers in

service (two servers that are busy). Therefore, knowledge of  $N(t)$  is sufficient to describe the state of the system. Because  $N(t) = N_q(t) + N_b(t)$  is true, the following relationship between expected values is also true.

$$E[N(t)] = E[N_q(t)] + E[N_b(t)] \quad (8.2)$$

If  $ST$  is the service time of an arbitrary customer, then it should be clear that

$$E[T] = E[T_q] + E[ST] \quad (8.3)$$

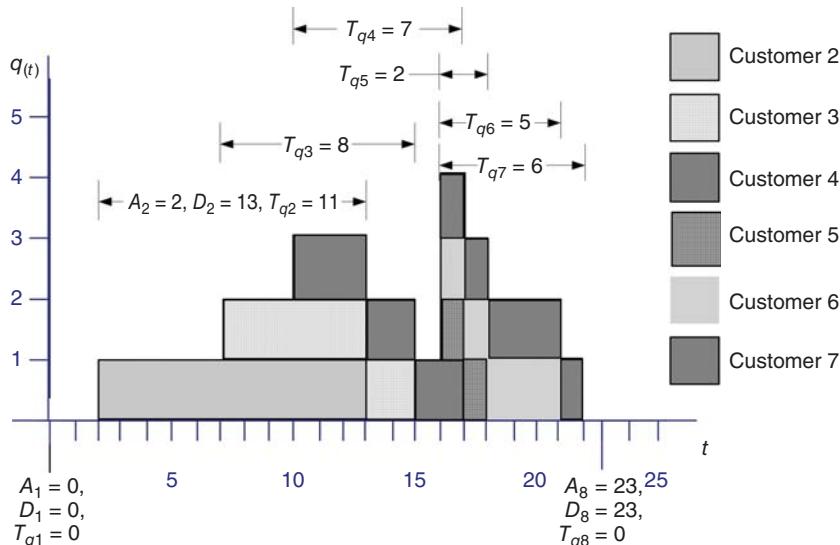
That is, the expected system time is equal to the expected waiting time in the queue plus the expected time spent in service.

### 8.2.2 Little's Formula

Chapter 7 presented time-persistent data and computed time averages. For queuing systems, one can show that relationships exist between such quantities as the expected number in the queue and the expected waiting time in the queue. Recall Figure 8.3 which illustrates the sample path for the number of customers in the queue over a period of time.

Let  $A_i; i = 1, \dots, n$  represent the time that the  $i$ th customer enters the queue,  $D_i; i = 1, \dots, n$  represent the time that the  $i$ th customer exits the queue, and  $T_{q_i} = D_i - A_i$  for  $i = 1, \dots, n$  represent the time that the  $i$ th customer spends in the queue. Recall that the average time spent in the queue was

$$\bar{T}_q = \frac{\sum_{i=1}^n T_{q_i}}{n} = \frac{0 + 11 + 8 + 7 + 2 + 5 + 6 + 0}{8} = \frac{39}{8} = 4.875$$



**Figure 8.3** Sample path for the number in a queue.

The average number of customers in the queue was

$$\begin{aligned}\bar{L}_q &= \frac{0(2-0) + 1(7-2) + 2(10-7) + 3(13-10) + 2(15-13) + 1(16-15)}{25} \\ &\quad + \frac{4(17-16) + 3(18-17) + 2(21-18) + 1(22-21) + 0(25-22)}{25} \\ &= \frac{39}{25} = 1.56\end{aligned}$$

By considering the waiting time lengths within the figure, the area under the sample path curve can be computed as

$$\sum_{i=1}^n T_{q_i} = 39$$

But, by definition, the area should also be

$$\int_{t_0}^{t_n} q(t) dt$$

Thus, it is no coincidence that the computed value for the numerators in  $\bar{T}_q$  and  $\bar{L}_q$  for the example is 39. Operationally, this must be the case.

Define  $\bar{R}$  as the average rate that customers exit the queue. The average rate of customer exiting the queue can be estimated by counting the number of customers who exit the queue over a period of time. That is,

$$\bar{R} = \frac{n}{t_n - t_0} \quad (8.4)$$

where  $n$  is the number of customers who departed the system during the time  $t_n - t_0$ . This quantity is often called the average throughput rate. For this example,  $\bar{R} = 8/25$ . By combining these equations, it becomes clear that the following relationship holds:

$$\bar{L}_q = \frac{\int_{t_0}^{t_n} q(t) dt}{t_n - t_0} = \frac{n}{t_n - t_0} \times \frac{\sum_{i=1}^n T_{q_i}}{n} = \bar{R} \times \bar{T}_q \quad (8.5)$$

This relationship is a conservation law and can also be applied to other portions of the queuing system as well. In words, the relationship states that

$$\text{Average number in queue} = \text{Average throughput rate} \times \text{Average waiting time in queue}$$

When the service portion of the system is considered, then the relationship can be translated as

$$\text{Average number in service} = \text{Average throughput rate} \times \text{Average time in service}$$

When the entire queuing system is considered, the relationship yields

$$\text{Average number in the system} = \text{Average throughput rate} \times \text{Average time in the system}$$

These relationships hold operationally for these statistical quantities as well as for the expected values of the random variables that underlie the stochastic processes. This relationship is called Little's formula after the queuing theorist who first formalized the technical conditions of its applicability to the stochastic processes within queues of this nature. The interested reader is referred to Little [1961] and Glynn and Whitt [1989] for more on these relationships. In particular, Little's formula states a relationship between the steady-state expected values for these processes.

To develop these formulas, define  $N$ ,  $N_q$ , and  $N_b$  as random variables that represent the number of customers in the system, in the queue, and in service at an arbitrary point in time in steady state. Also, let  $\lambda$  be the expected arrival rate so that  $1/\lambda$  is the mean of the interarrival time distribution and let  $\mu = 1/E[ST]$  so that  $E[ST] = 1/\mu$  is the mean of the service time distribution. The expected values of the quantities of interest can be defined as

$$L \equiv E[N]$$

$$L_q \equiv E[N_q]$$

$$B \equiv E[N_b]$$

$$W \equiv E[T]$$

$$W_q \equiv E[T_q]$$

Thus, it should be clear that

$$L = L_q + B \quad (8.6)$$

$$W = W_q + E[ST] \quad (8.7)$$

In steady state, the mean arrival rate to the system should also be equal to the mean throughput rate. Thus, from Little's relationship, the following are true:

$$L = \lambda W \quad (8.8)$$

$$L_q = \lambda W_q \quad (8.9)$$

$$B = \lambda E[ST] = \frac{\lambda}{\mu} \quad (8.10)$$

To gain an intuitive understanding of Little's formulas in this situation, consider that in steady state, the mean rate that customers exit the queue must also be equal to the mean rate that customers enter the queue. Suppose that you are a customer who is departing the queue and you look behind yourself to see how many customers are left in the queue. This quantity should be  $L_q$  on average. If it took you on average  $W_q$  to get through the queue, how many customers would have arrived on average during this time? If the customers arrive at rate  $\lambda$ , then  $\lambda \times W_q$  is the number of customers (on average) that would have arrived during your time in the queue, but these are the customers who you would see (on average) when looking behind you. Thus,  $L_q = \lambda W_q$ .

Notice that  $\lambda$  and  $\mu$  must be given and, therefore,  $B$  is known. The quantity  $B$  represents the expected number of customers in service in steady state, but since a customer uses only one server while in service,  $B$  also represents the expected number of busy servers in steady state. If there are  $c$  identical servers in the resource, then the quantity,  $B/c$  represents the

fraction of the servers that are busy. This quantity can be interpreted as the utilization of the resource as a whole or the average utilization of a server, since they are all identical. This quantity is defined as

$$\rho = \frac{B}{c} = \frac{\lambda}{c\mu} \quad (8.11)$$

The quantity,  $c\mu$ , represents the maximum rate at which the system can perform work on average. Because of this,  $c\mu$  can be interpreted as the mean capacity of the system. One of the technical conditions that is required for Little's formula to be applicable is that  $\rho < 1$  or  $\lambda < c\mu$ . That is, the mean arrival rate to the system must be less than mean capacity of the system. This also implies that the utilization of the resource must be less than 100%.

The queuing system can also be characterized in terms of the *offered load*. The offered load is a dimensionless quantity that gives the average amount of work offered per time unit to the  $c$  servers. The offered load is defined as  $r = \lambda/\mu$ . Notice that this can be interpreted as each customer arriving with  $1/\mu$  average units of work to be performed. The steady-state conditions thus indicate that  $r < c$ . In other words, the arriving amount of work to the queue cannot exceed the number of servers. These conditions make sense for steady-state results to be applicable, since if the mean arrival rate was greater than the mean capacity of the system, the waiting line would continue to grow over time.

### 8.2.3 Deriving Formulas for Markovian Single-Queue Systems

Notice that with  $L = \lambda W$  and the other relationships, all of the major performance measures for the queue can be computed if a formula for one of the major performance measures (e.g.,  $L$ ,  $L_q$ ,  $W$ , or  $W_q$ ) is available. In order to derive formulas for these performance measures, the arrival and service processes must be specified.

This section shows that for the case of exponential time between arrivals and exponential service times, the necessary formulas can be derived. It is useful to go through the basic derivations in order to better understand the interpretation of the various performance measures, the implications of the assumptions, and the concept of steady state.

To motivate the development, let us consider a simple example. Suppose you want to model an old style telephone booth, which can hold only one person while the person uses the phone. Also assume that any people who arrive while the booth is in use immediately leave. In other words, nobody waits to use the booth.

For this system, it is important to understand the behavior of the stochastic process  $N(t); t \geq 0$ , where  $N(t)$  represents the number of people who are in the phone booth at any time  $t$ . Clearly, the possible values of  $N(t)$  are 0 and 1, that is,  $N(t) \in \{0, 1\}$ . Developing formulas for the probability that there are 0 or 1 customers in the booth at any time  $t$ , that is,  $P_i(t) = P\{N(t) = i\}$  will be the key to modeling this situation.

Let  $\lambda$  be the mean arrival rate of customers to the booth and let  $ST = 1/\mu$  be the expected length of a telephone call. For example, if the mean time between arrivals is 12 minutes, then  $\lambda = 5$  per hour, and if the mean length of a call is 10 minutes, then  $\mu = 6$  per hour. The following reasonable assumptions will be made:

1. The probability of a customer arriving in a small interval of time,  $\Delta t$ , is roughly proportional to the length of the interval, with the proportionality constant equal to the mean rate of arrival,  $\lambda$ .

2. The probability of a customer completing an ongoing phone call during a small interval of time is roughly proportional to the length of the interval and the proportionality constant is equal to the mean service rate,  $\mu$ .
3. The probability of more than one arrival in an arbitrarily small interval,  $\Delta t$ , is negligible. In other words,  $\Delta t$ , can be made small enough so that only one arrival can occur in the interval.
4. The probability of more than one service completion in an arbitrarily small interval,  $\Delta t$ , is negligible. In other words,  $\Delta t$ , can be made small enough so that only one service can occur in the interval.

Let  $P_0(t)$  and  $P_1(t)$  represent the probability that there is 0 or 1 customer using the booth, respectively. Suppose that you observe the system at time  $t$  and you want to derive the probability for 0 customers in the system at some future time,  $t + \Delta t$ . Thus, you want  $P_0(t + \Delta t)$ .

For there to be zero customers in the booth at time  $t + \Delta t$ , there are two possible situations that could occur. First, there could have been no customers in the system at time  $t$  and no arrivals during the interval  $\Delta t$ , or there could have been one customer in the system at time  $t$  and the customer completed service during  $\Delta t$ . Thus, the following relationship should hold:

$$\begin{aligned} \{N(t + \Delta t) = 0\} &= \{\{N(t) = 0\} \cap \{\text{no arrivals during } \Delta t\}\} \cup \\ &\quad \{\{N(t) = 1\} \cap \{\text{service completed during } \Delta t\}\} \end{aligned}$$

It follows that

$$P_0(t + \Delta t) = P_0(t)P\{\text{no arrivals during } \Delta t\} + P_1(t)P\{\text{service completed during } \Delta t\}$$

In addition, at time  $t + \Delta t$ , there might be a customer using the booth,  $P_1(t + \Delta t)$ . For there to be one customer in the booth at time  $t + \Delta t$ , there are two possible situations that could occur. First, there could have been no customers in the system at time  $t$  and one arrival during the interval  $\Delta t$ , or there could have been one customer in the system at time  $t$  and the customer did not complete service during  $\Delta t$ . Thus, the following holds:

$$P_1(t + \Delta t) = P_0(t)P\{1 \text{ arrival during } \Delta t\} + P_1(t)P\{\text{service completed during } \Delta t\}$$

Because of assumptions (1) and (2), the following probability statements can be used:

$$\begin{aligned} P\{1 \text{ arrival during } \Delta t\} &\cong \lambda \Delta t \\ P\{\text{no arrivals during } \Delta t\} &\cong 1 - \lambda \Delta t \\ P\{\text{service completed during } \Delta t\} &\cong \mu \Delta t \\ P\{\text{service completed during } \Delta t\} &\cong 1 - \mu \Delta t \end{aligned}$$

This results in the following:

$$P_0(t + \Delta t) = P_0(t)[1 - \lambda \Delta t] + P_1(t)[\mu \Delta t] \tag{8.12}$$

$$P_1(t + \Delta t) = P_0(t)[\lambda \Delta t] + P_1(t)[1 - \mu \Delta t] \tag{8.13}$$

Collecting the terms in the equations, rearranging, dividing by  $\Delta t$ , and taking the limit as goes  $\Delta t$  to zero yields the following set of differential equations:

$$\frac{dP_0(t)}{dt} = \lim_{\Delta t \rightarrow 0} \frac{P_0(t + \Delta t) - P_0(t)}{\Delta t} = -\lambda P_0(t) + \mu P_1(t) \quad (8.14)$$

$$\frac{dP_1(t)}{dt} = \lim_{\Delta t \rightarrow 0} \frac{P_1(t + \Delta t) - P_1(t)}{\Delta t} = \lambda P_0(t) - \mu P_1(t) \quad (8.15)$$

It is also true that  $P_0(t) + P_1(t) = 1$ . Assuming that  $P_0(0) = 1$  and  $P_1(0) = 0$  as the initial conditions, the solutions to these differential equations are

$$P_0(t) = \left( \frac{\mu}{\lambda + \mu} \right) + \left( \frac{\lambda}{\lambda + \mu} \right) e^{-(\lambda+\mu)t} \quad (8.16)$$

$$P_1(t) = \left( \frac{\lambda}{\lambda + \mu} \right) - \left( \frac{\lambda}{\lambda + \mu} \right) e^{-(\lambda+\mu)t} \quad (8.17)$$

These equations represent the probability of having either 0 or 1 customer in the booth at any time. If the limit as  $t$  goes to infinity is considered, the *steady-state probabilities*,  $P_0$  and  $P_1$ , can be determined:

$$P_0 = \lim_{t \rightarrow \infty} P_0(t) = \frac{\lambda}{\lambda + \mu} \quad (8.18)$$

$$P_1 = \lim_{t \rightarrow \infty} P_1(t) = \frac{\mu}{\lambda + \mu} \quad (8.19)$$

These probabilities can be interpreted as the chance that an arbitrary customer finds the booth either empty or busy after an infinitely long period of time has elapsed.

If only the steady-state probabilities are desired, there is an easier method to perform the derivation, from both a conceptual and a mathematical standpoint. The assumptions (1–4) that were made ensure that the arrival and service processes will be Markovian. In other words, the time between arrivals of the customer is exponentially distributed and the service times are exponentially distributed. In addition, the concept of steady state can be used.

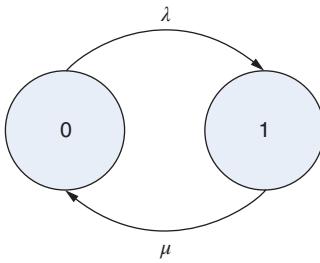
Consider the differential equations. These equations govern the rate of change of the *probabilities* over time. Consider the analogy of water to probability and think of a dam or container that holds an amount of water. The rate of change of the level of water in the container can be thought of as

$$\text{Rate of change of level} = \text{Rate into container} - \text{Rate out of the container}$$

In steady state, the level of the water should not change, thus the rate into the container must equal the rate out of the container. Using this analogy,

$$\frac{dP_i(t)}{dt} = \text{Rate in} - \text{Rate out}$$

and for steady state: *rate in = rate out* with the probability *flowing* between the states. Figure 8.4 illustrates this concept via a state-transition diagram. If  $N$  represents the steady-state number of customers in the system (booth), the two possible states that the

**Figure 8.4** Two-state-transition diagram.

system can be in are 0 and 1. The rate of transition from state 0 to state 1 is the rate that an arrival occurs and the state of transition from state 1 to state 0 is the service rate.

The rate of transition into state 0 can be thought of as the rate that probability flows from state 1 to state 0 times the chance of being in state 1, that is,  $\mu P_1$ . The rate of transition out of state 0 can be thought of as the rate from state 0 to state 1 times the chance of being in state 0, that is,  $\lambda P_0$ . Using these ideas yields

<i>State</i>	<i>Rate in</i>	=	<i>Rate out</i>
0	$\mu P_1$	=	$\lambda P_0$
1	$\lambda P_0$	=	$\mu P_1$

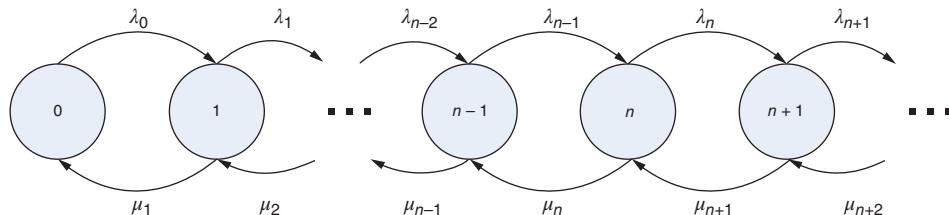
Notice that these are identical equations, but with the fact that  $P_0 + P_1 = 1$ , we will have two equations and two unknowns ( $P_0, P_1$ ). Thus, the equations can be easily solved to yield the same results as Equations (8.18) and (8.19). Sets of equations derived in this manner are called steady-state equations.

Now, more general situations can be examined. Consider a general queuing system with some given number of servers. An arrival to the system represents an increase in the number of customers and a departure from the system represents a decrease in the number of customers in the system.

Figure 8.5 illustrates a general state-transition diagram for this system. Let  $N$  be the number of customers in the system in steady state and define

$$P_n = P\{N = n\} = \lim_{t \rightarrow \infty} P\{N(t) = n\} \quad (8.20)$$

as the steady-state probability that there are  $n$  customers in the system. Let  $\lambda_n$  be the mean arrival rate of customers entering the system when there are  $n$  customers in the system,

**Figure 8.5** General state-transition diagram.

$\lambda_n \geq 0$ . Let  $\mu_n$  be the mean service rate for the overall system when there are  $n$  customers in the system. This is the rate, at which customers depart when there are  $n$  customers in the system. In this situation, the number of customers may be infinite, that is,  $N \in \{0, 1, 2, \dots\}$ . The steady-state equations for this situation are as follows:

<i>State</i>	<i>Rate in</i>	=	<i>Rate out</i>
0	$\mu_1 P_1$	=	$\lambda_0 P_0$
1	$\lambda_0 P_0 + \mu_2 P_2$	=	$\mu_1 P_1 + \lambda_1 P_1$
2	$\lambda_1 P_1 + \mu_3 P_3$	=	$\mu_2 P_2 + \lambda_2 P_2$
⋮	⋮	⋮	⋮
$n$	$\lambda_{n-1} P_{n-1} + \mu_{n+1} P_{n+1}$	=	$\mu_n P_n + \lambda_n P_n$

These equations can be solved recursively starting with state 0. This yields

$$\begin{aligned} P_1 &= \frac{\lambda_0}{\mu_1} P_0 \\ P_2 &= \frac{\lambda_1 \lambda_0}{\mu_2 \mu_1} P_0 \\ &\vdots \\ P_n &= \frac{\lambda_{n-1} \lambda_{n-2} \cdots \lambda_0}{\mu_n \mu_{n-1} \cdots \mu_1} P_0 = \prod_{j=1}^{n-1} \left( \frac{\lambda_j}{\mu_{j+1}} \right) P_0 \end{aligned}$$

for  $n = 1, 2, 3, \dots$  Provided that  $\sum_{n=0}^{\infty} P_n = 1$ ,  $P_0$  can be computed as

$$P_0 = \left[ \sum_{n=0}^{\infty} \prod_{j=0}^{n-1} \left( \frac{\lambda_j}{\mu_{j+1}} \right) \right]^{-1} \quad (8.21)$$

Therefore, for any given set of  $\lambda_n$  and  $\mu_n$ , one can compute  $P_n$ . The  $P_n$  represent the steady-state probabilities of having  $n$  customers in the system. Because  $P_n$  is a probability distribution, the expected value of this distribution can be computed. What is the expected value for the  $P_n$  distribution? The expected number of customers in the system in steady state. This is  $L$ . The expected number of customers in the system and the queue are given by

$$L = \sum_{n=0}^{\infty} n P_n \quad (8.22)$$

$$L_q = \sum_{n=c}^{\infty} (n - c) P_n \quad (8.23)$$

where  $c$  is the number of servers.

There is one additional formula that is needed before Little's formula can be applied with other known relationships. For certain systems, for example, finite system size, not all

customers who arrive will enter the queue. Little's formula is true for the customers who enter the system. Thus, the effective arrival rate must be defined. The effective arrival rate is the mean rate of arrivals that actually enter the system. This is given by computing the expected arrival rate across the states. For infinite system size, we have

$$\lambda_e = \sum_{n=0}^{\infty} \lambda_n P_n \quad (8.24)$$

For finite system size,  $k$ , we have

$$\lambda_e = \sum_{n=0}^{k-1} \lambda_n P_n \quad (8.25)$$

since  $\lambda_n = 0$  for  $n \geq k$ . This is because nobody can enter when the system is full. All these relationships yield

$$L = \lambda_e W \quad (8.26)$$

$$L_q = \lambda_e W_q \quad (8.27)$$

$$B = \frac{\lambda_e}{\mu} \quad (8.28)$$

$$\rho = \frac{\lambda_e}{c\mu} \quad (8.29)$$

$$L = L_q + B \quad (8.30)$$

$$W = W_q + \frac{1}{\mu} \quad (8.31)$$

Appendix D presents the results of applying the general solution for  $P_n$  to different queuing system configurations. Table 8.1 presents specific results for the M/M/ $c$  queuing system for  $c = 1, 2, 3$ . Using these results and those in Appendix D, the analysis of a variety of different queuing situations is possible.

This section has only scratched the surface of queuing theory. A vast amount of literature is available on queuing theory and its application. You should examine Gross and Harris [1998], Cooper [1990], and Kleinrock [1975] for a more in-depth theoretical development of the topic. There are also a number of free online resources available on the topic. The interested reader should search on “Queueing Theory Books On Line.”

**TABLE 8.1 Results M/M/ $c$   $\rho = \lambda/c\mu$**

$c$	$P_0$	$L_q$
1	$1 - \rho$	$\frac{\rho^2}{1 - \rho}$
2	$\frac{1 - \rho}{1 + \rho}$	$\frac{2\rho^3}{1 - \rho^2}$
3	$\frac{2(1 - \rho)}{2 + 4\rho + 3\rho^2}$	$\frac{9\rho^4}{2 + 2\rho - \rho^2 - 3\rho^3}$

### 8.3 EXAMPLES AND APPLICATIONS OF QUEUING ANALYSIS

The derivations and formulas in the previous section certainly appear to be intimidating and they can be tedious to apply. Fortunately, there is readily available software that can be used to do the calculations. Online resources for software for queuing analysis can be found at

- List of queuing theory software resources.<sup>1</sup>
- QTSPlus<sup>2</sup> Excel-based queuing theory software that accompanies Gross et al. [2008].

The most important part of performing a queuing analysis is to identify the most appropriate queuing model for a given situation. Then, software tools can be used to analyze the situation. This section provides a number of examples and discusses the differences between the systems so that you can better apply the results of the previous sections. The solutions to these types of problems involve the following steps:

1. Identify the arrival and service processes.
2. Identify the size of the arriving population and the size of the system.
3. Specify the appropriate queuing model and its input parameters.
4. Identify the desired performance measures.
5. Compute the required performance measures.

#### 8.3.1 Infinite Queue Examples

In this section, we explore two queuing systems ( $M/M/1$  and  $M/M/c$ ) that have an infinite population of arrivals and an infinite size queue. The examples illustrate some of the common questions related to these types of queuing systems.

##### EXAMPLE 8.1 Drive-Through Pharmacy

Customers arrive at a one-window drive-through pharmacy according to a Poisson distribution with a mean of 10 per hour. The service time per customer is exponential with a mean of 5 minutes. There are three spaces in front of the window, including that for the car being served. Other arriving cars can wait outside these three spaces. The pharmacy is interested in answering the following questions:

- (a) What is the probability that an arriving customer can enter one of the three spaces in front of the window?
- (b) What is the probability that an arriving customer will have to wait outside the three spaces?
- (c) What is the probability that an arriving customer has to wait?
- (d) How long is an arriving customer expected to wait before starting service?
- (e) How many car spaces should be provided in front of the window so that an arriving customer has a  $\gamma = 40\%$  chance of being able to wait in one of the provided spaces?

<sup>1</sup><http://web2.uwindsor.ca/math/hlynka/qsoft.html>.

<sup>2</sup><http://mason.gmu.edu/jshortle/fqt4th.html>.

*Solution to Example 8.1:* The customers arrive according to a Poisson process, which implies that the time between arrivals is exponentially distributed. Thus, the arrival process is Markovian (M). The service process is stated as exponential. Thus, the service process is Markovian (M). There is only one window and customers wait in front of this window to receive service. Thus, the number of servers is  $c = 1$ . The problem states that customers who arrive when the three spaces are filled, still wait for service outside the three spaces. Thus, there does not appear to be a restriction on the size of the waiting line. Therefore, this situation can be considered an infinite size system. The arrival rate is specified for any likely customer and there is no information given concerning the total population of the customers. Thus, it appears that an infinite population of customers can be assumed. We can conclude that this is an M/M/1 queuing situation with an arrival rate  $\lambda = 10$  per hour and a service rate of  $\mu = 12$  per hour. Notice the input parameters have been converted to a common unit of measure (customers per hour).

The equations for the M/M/1 can be readily applied. From Appendix D, the following formulas can be applied:

$$c = 1$$

$$\lambda_n = \lambda = 10 \text{ per hour}$$

$$\lambda_e = \lambda$$

$$\mu_n = \mu = 12 \text{ per hour}$$

$$\rho = \frac{\lambda}{c\mu} = r = 10/12 = 5/6$$

$$P_0 = 1 - \frac{\lambda}{\mu} = 1 - r = 1/6$$

$$P_n = P_0 r^n = \frac{1}{6} \left(\frac{5}{6}\right)^n$$

$$L = \frac{r}{1-r} = \frac{(5/6)}{1-(5/6)} = 5$$

$$L_q = \frac{r^2}{1-r} = \frac{(5/6)^2}{1-(5/6)} = 4.1\bar{6}$$

$$W_q = \frac{L_q}{\lambda} = \frac{r}{\mu(1-r)} = 0.41\bar{6} \text{ hours} = 25.02 \text{ minutes}$$

$$W = \frac{L}{\lambda} = \frac{5}{10} = 0.5 \text{ hours} = 30 \text{ minutes}$$

Let us consider each question in turn:

- (a) Probability statements of this form are related to the underlying state variable for the system. In this case, let  $N$  represent the number of customers in the system. To find the probability that an arriving customer can enter one of the three spaces in front of the window, you should consider the question: *When can a customer enter one of the three spaces?* A customer can enter one of the three spaces when there are 0 or 1 or 2 customers in the system. This is not  $N = 0, 1, 2$ , or 3 because if there are three

customers in the system, then the third space is taken. Therefore,  $P\{N \leq 2\}$  needs to be computed.

To compute  $P\{N \leq 2\}$ , note that

$$P\{N \geq n\} = \sum_{j=n}^{\infty} P_0 r^j = (1-r) \sum_{j=n}^{\infty} r^{j-n} = (1-r) \frac{r^n}{1-r} = r^n$$

Therefore,  $P\{N \leq n\}$  is

$$P\{N \leq n\} = 1 - P\{N > n\} = 1 - P\{N \geq n+1\} = 1 - r^{n+1}$$

Thus, we have that  $P\{N \leq 2\} = 1 - r^3 \cong 0.42$

- (b) An arriving customer will have to wait outside of the three spaces, when there are more than 2 (3 or more) customers in the system. Thus,  $P\{N > 2\}$  needs to be computed to answer question (b). This is the complement event for part (a).

$$P\{N > 2\} = 1 - P\{N \leq 2\} \cong 0.58$$

- (c) An arriving customer has to wait when there are one or more customers already at the pharmacy. This is  $P\{N \geq 1\} = 1 - P\{N < 1\} = 1 - P_0$ .

$$P\{N \geq 1\} = 1 - P\{N < 1\} = 1 - P_0 = 1 - (1-r) = \rho = \frac{5}{6}$$

- (d) The waiting time that does not include service is the queuing time. Thus,  $W_q$  needs to be computed to answer question (d).

$$W_q = 0.41\bar{6} \text{ hours} = 25.02 \text{ minutes}$$

- (e) This is a design question for which the probability of waiting in one of the provided spaces is used to determine the number of spaces to provide. Suppose that there are  $m$  spaces. An arriving customer can wait in one of the spaces if there are  $m-1$  or less customers in the system. Thus,  $m$  needs to be chosen such that  $P\{N \leq m-1\} = 0.4$ .

$$P\{N \leq m-1\} = \gamma$$

$$1 - r^m = \gamma$$

$$r^m = 1 - \gamma$$

$$m = \frac{\ln(1-\gamma)}{\ln r} = \frac{\ln(1-0.4)}{\ln(\frac{5}{6})} = 2.8 \cong 3 \text{ spaces}$$

Rounding up guarantees  $P\{N \leq m-1\} \geq 0.4$ .

## EXAMPLE 8.2 Self-Service Copiers

The Student Union Copy center is considering the installation of self-service copiers. They predict that the arrivals will be Poisson with a rate of 30 per hour and that the time spent copying is exponentially distributed with a mean of 1.75 minutes. They would

like the chance that four or more people in the copy center to be less than 5%. How many copiers should they install?

*Solution to Example 8.2:* The Poisson arrivals and exponential service times make this situation an M/M/c where  $c$  is the number of copiers to install and  $\lambda = 0.5$  and  $\mu = 1/1.75$  per minute. To meet the design criteria,  $P\{N \geq 4\} = 1 - P\{N \leq 3\}$  needs to be computed for systems with  $c = 1, 2, \dots$  until  $P\{N \geq 4\}$  is less than 5%. This can be readily achieved with the provided formulas or by utilizing the aforementioned software. In what follows, the QTSPlus<sup>3</sup> software was used.

The software is very self-explanatory. It is important to remember that when applying the queuing formulas that you keep your time units consistent. Setting up the QTSPlus spreadsheet as shown in Figure 8.6 with  $c = 3$  yields the results shown in Figure 8.7. By changing the number of servers, one can find that  $c = 3$  meets the probability requirement as shown in Table 8.2.

**8.3.1.1 Square Root Staffing Rule** Often in the case of systems with multiple servers such as the M/M/c, you want to determine the best value of  $c$ , as in Example 8.2. Another common design situation is to determine the value of  $c$  such that there is an acceptable probability that an arriving customer will have to wait. For the case of Poisson arrivals, this is the same as the steady-state probability that there are more than  $c$  customers in the system. For the M/M/c model, this is called the Erlang delay probability:

$$P_w = P\{N \geq c\} = \sum_{n=c}^{\infty} P_n = \frac{\frac{r^c}{c!}}{\frac{r^c}{c!} + (1 - \rho) \sum_{j=0}^{c-1} \frac{r^j}{j!}} \quad (8.32)$$

Even though this is a relatively easy formula to use (especially in view of available spreadsheet software), an interesting and useful approximation has been developed called the square root staffing rule. The derivation of the square root staffing rule is given in Tijms [2003]. In what follows, the usefulness of the rule is discussed.

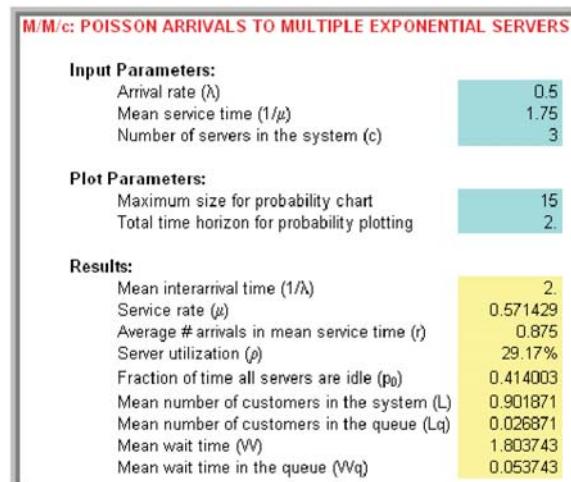


Figure 8.6 QTSPlus M/M/c spreadsheet.

<sup>3</sup><http://mason.gmu.edu/~jshortle/fqt4th.html>.

	A	B	C	D
1	<b>Customer Size Distribution</b>			
n	prob(n)	CDF(n)	1-CDF(n)	
0	0.414003	0.414003	0.585997	
1	0.362253	0.776256	0.223744	
2	0.158486	0.934741	0.065259	
3	0.046225	0.980966	0.019034	
4	0.013482	0.994448	0.005552	
5	0.003932	0.998381	0.001619	
6	0.001147	0.999528	0.000472	
7	0.000335	0.999862	0.000138	
8	0.000098	0.999960	0.000040	
9	0.000028	0.999988	0.000012	
10	0.000008	0.999997	0.000003	
11	0.000002	0.999999	0.000001	
12	0.000001	1.000000	0.000000	
13	0.000000	1.000000	0.000000	
14	0.000000	1.000000	0.000000	
15	0.000000	1.000000	0.000000	
16	0.000000	1.000000	0.000000	
17	0.000000	1.000000	0.000000	
18	0.000000	1.000000	0.000000	

**Figure 8.7** QTSPlus results for Example 8.2 for  $c = 3$ .**TABLE 8.2** Results for Example 8.2,  
 $c = 1, 2, 3, 4$ 

$c$	$P\{N \geq 4\} = 1 - P\{N \leq 3\}$
1	0.586182
2	0.050972
3	0.019034
4	0.013022

The *square root staffing rule* states that the least number of servers,  $c^*$ , required to meet the criteria  $P_w \leq \alpha$  is given by  $c^* \cong r + \gamma_\alpha \sqrt{r}$ , where the factor  $\gamma_\alpha$  is the solution to the equation

$$\frac{\gamma \Phi(\gamma)}{\varphi(\gamma)} = \frac{1 - \alpha}{\alpha} \quad (8.33)$$

The functions  $\Phi(\cdot)$  and  $\varphi(\cdot)$  are the cumulative distribution function (CDF) and the probability density function (PDF) of a standard normal random variable. Therefore, given a design criteria,  $\alpha$ , in the form a probability tolerance, you can find the number of servers that will result in the probability of waiting being less than  $\alpha$ . This has very useful application in the area of call service centers and help support lines.

### ■ EXAMPLE 8.3 Square Root Staffing Rule

The Student Union Copy center is considering the installation of self-service copiers. They predict that the arrivals will be Poisson with a rate of 30 per hour and that the time spent copying is exponentially distributed with a mean of 1.75 minutes. Find the least number of servers such that the probability that an arriving customer waits is less than or equal to 0.10.

	A	B	C	D	E
1	Square Root Staffing Rule				
2					
3	Offered Load = r	0.875			
4	Probability of delay criteria = alpha	0.1			
5	Initial Factor = gamma	1.420189439			
6	LHS of factor equation	9.000052301	$\gamma \Phi(r) = \frac{1-\alpha}{\alpha}$		
7	RHS of factor equation	9	$\varphi(r)$		
8	Root finding equation	-5.23013E-05			
9	Approximate number of servers needed	2.203465572	$c \approx r + \gamma \sqrt{r}$		
10	Number of servers rounded up	3			
11					
12	Goal Seek				
13	Set cell:	B8			
14	To value:	0			
15	By changing cell:	\$B\$5			
16			OK	Cancel	
17					
18					
19					
20					
21					
22					

1) Enter the offered load and delay criteria  
 2) Enter an initial value for gamma. The value 1 will always work.  
 3) Run goal seek as shown (setting B8 to zero by changing B5)

Once the factor gamma has been found for the given alpha, you can vary the offered load and get the number of servers needed for different values of the offered load.

Figure 8.8 Spreadsheet for square root staffing rule.

*Solution to Example 8.3:* For  $\lambda = 0.5$  and  $\mu = 1/1.75$  per minute, you have that  $r = 0.875$ . Figure 8.8 illustrates the use of the *SquareRootStaffingRule.xls* spreadsheet that accompanies this chapter. The spreadsheet has text that explains the required inputs. Enter the offered load  $r$  in cell B3 and the delay criteria of 0.1 in cell B4. An initial search value is required in cell B5. The value of 1.0 will always work. The spreadsheet uses the goal seek functionality to solve for  $\gamma_\alpha$ .

For this problem, this results in about three servers being needed to ensure that the probability of wait will be less than 10%. The square root staffing rule has been shown to be quite a robust approximation and can be useful in many design settings involving staffing.

### 8.3.2 Finite Queue Examples

In this section, we explore three queuing systems that are finite in some manner, either in space or in population. The examples illustrate some of the common questions related to these types of systems.

#### EXAMPLE 8.4 A Finite Buffer Queue

A single machine is connected to a conveyor system. The conveyor causes parts to arrive to the machine at a rate of 1 part per minute according to a Poisson distribution. There is a finite buffer of size 5 in front of the machine. The machine's processing time is considered to be exponentially distributed with a mean rate of 1.2 parts per minute. Any parts that arrive on the conveyor when the buffer is full are carried to other machines that are not part of this analysis. What are the expected system time and the expected number of parts at the machining center?

*Solution to Example 8.4:* The finite buffer, Poisson arrivals, and exponential service times make this situation an M/M/1/6 where  $k = 6$  is the size of the system (5 in buffer + 1 in

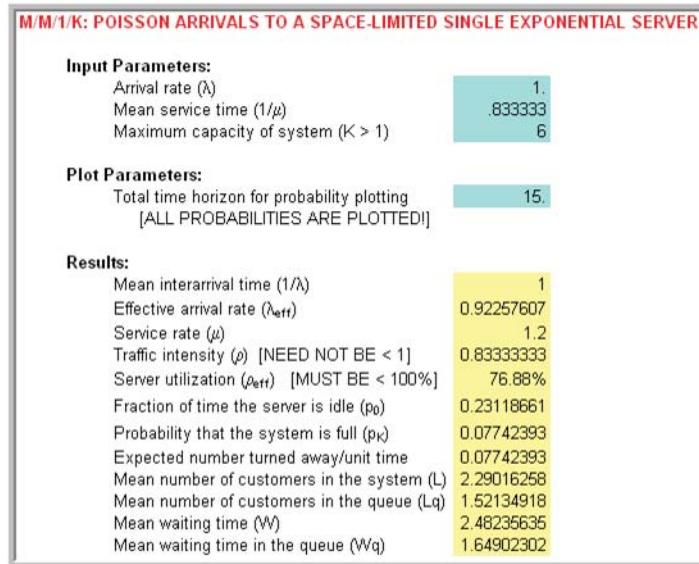


Figure 8.9 Results for Example 8.4.

service) and  $\lambda = 1$  and  $\mu = 1.2$  per minute. The desired performance measures are  $W$  and  $L$ . Figure 8.9 presents the results using *QTSPplus*. Notice that in this case, the effective arrival rate must be computed:

$$\lambda_e = \sum_{n=0}^{k-1} \lambda_n P_n = \sum_{n=0}^{k-1} \lambda P_n = \lambda \sum_{n=0}^{k-1} P_n = \lambda(1 - P_k) \quad (8.34)$$

Rearranging this formula yields  $\lambda = \lambda_e + \lambda_f$  where  $\lambda_f = \lambda P_k$  equals the mean number of customers who are turned away from the system because it is full. According to Figure 8.9, the expected number of customer turned away because the system is full is about 0.077 per minute (or about 4.62 per hour).

## ■ EXAMPLE 8.5 A Parking Lot

The university has a row of 10 parking meter based spaces across from the engineering school. During the peak hours, students arrive to the parking lot at a rate of 40 per hour according to a Poisson distribution and the students use the parking space for approximately 60 minutes exponentially distributed. If all the parking spaces are taken, it can be assumed that an arriving student does not wait (goes somewhere else to park). Suppose that the meters cost  $w = \$0.03$  per minute, that is,  $\$2$  per hour. How much income does the university potentially lose during peak hours on an average because the parking spaces are full?

*Solution to Example 8.5:* In this system, the parking spaces are the servers of the system. There are 10 parking spaces so that  $c = 10$ . In addition, there is no waiting for one of the meters and thus no queue forms. Therefore, the system size,  $k$ , is also 10. Because of the Poisson arrival process and exponential service times, this can be considered an M/M/10/10

queueing system. In other words, the size of the system is same as the number of servers,  $c = k = 10$ .

In the case of an  $M/M/c/c$  queueing system,  $P_c$  represents the probability that all the servers in the system are busy. Thus, it also represents the probability that an arriving customer will be turned away. The formula for the probability of a lost customer is called the Erlang loss formula:

$$P_c = \frac{\frac{r^c}{c!}}{\sum_{n=0}^c \frac{r^n}{n!}} \quad (8.35)$$

Customers arrive at the rate  $\lambda = 20$  per hour whether the system is full or not. Thus, the expected number of lost customers per hour is  $\lambda P_c$ . Since each customer brings  $1/\mu$  service time charged at  $w = \$2$  per hour, each arriving customer brings  $w/\mu$  of income on an average; however, not all arriving customers can park. Thus, the university loses  $w \times 1/\mu \times \lambda P_c$  of income per hour. Figure 8.10 illustrates the use of the *QTSPPlus* software. In the figure, the arrival rate of 40 per hour and the mean service time of 1 hour is entered. The number of parking spaces, 10, is the size of the system.

Thus according to Figure 8.10, the university is losing about  $\$2 \times 30.3 = \$60.6$  per hour because the metered lot is full during peak hours.

For this example, the service times are exponentially distributed. It turns out that for the case of  $c = k$ , the results for the  $M/M/c/c$  model are the same for the  $M/G/c/c$  model. In other words, the form of the service distribution does not matter. The mean of the service distribution is critical to this analysis.

In the next example, a *finite population* of customers who can arrive, depart, and then return is considered. A classic and important example of this type of system is the machine interference or operator tending problem. A detailed discussion of the analysis and application of this model can be found in Stecke [1992].

In this situation, a set of machines are tended by one or more operators. The operators must tend to stoppages (breakdowns) of the machines. As the machines breakdown, they may have to wait for the operator to complete the service of other machines. Thus,

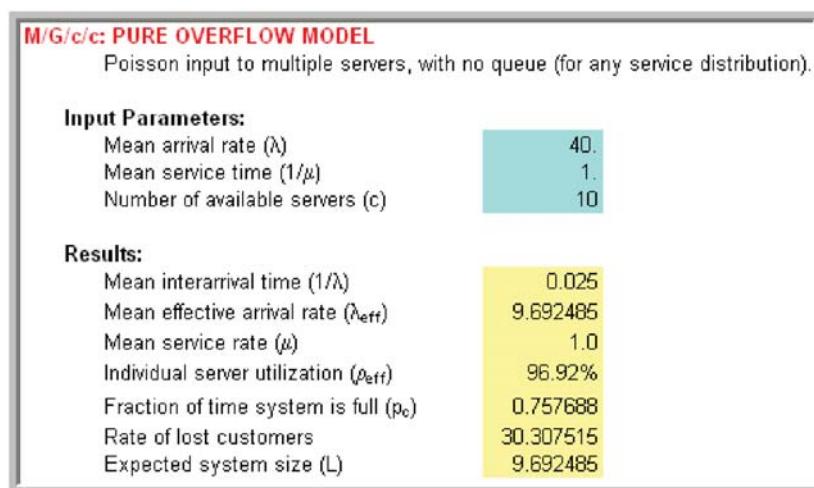


Figure 8.10 Example 8.5 M/M/c/c results.

the machine stoppages cause the machines to *interfere* with the productivity of the set of machines because of their dependence on a common resource, the operator.

The situation that we will examine and for which analytical formulas can be derived is called the  $M/M/c/k/k$  queuing model where  $c$  is the number of servers (operators) and  $k$  is the number of machines (size of the population). Notice that the size of the system is the same as the size of the calling population in this particular model. For this system, the arrival rate of an *individual machine*,  $\lambda$ , is specified. This is not the arrival rate of the population as has been previously utilized. The service rate of  $\mu$  for each operator is also necessary. The arrival and service rates for this system are

$$\lambda_n = \begin{cases} (k - n)\lambda & n = 0, 1, 2, \dots, k \\ 0 & n \geq k \end{cases} \quad (8.36)$$

$$\mu_n = \begin{cases} n\mu & n = 1, 2, \dots, c \\ c\mu & n \geq c \end{cases} \quad (8.37)$$

These rates are illustrated in the state diagram of Figure 8.11 for the case of two operators and five machines. Thus, for this system, the arrival rate to the system decreases as more machines breakdown. Notice that in the figure, the arrival rate from state 0 to state 1 is  $5\lambda$ . This is because there are five machines that are not broken down, each with individual rate,  $\lambda$ . Thus, the total rate of arrivals to the system is  $5\lambda$ . This rate goes down as more machines breakdown. When all the machines are broken down, the arrival rate to the system will be zero.

Notice also that the service rate from state 1 to state 0 is  $\mu$ . When one machine is in the system, the operator works at rate  $\mu$ . Note also that the rate from state 2 to state 1 is  $2\mu$ . This is because when there are two broken down machines, the two operators are working

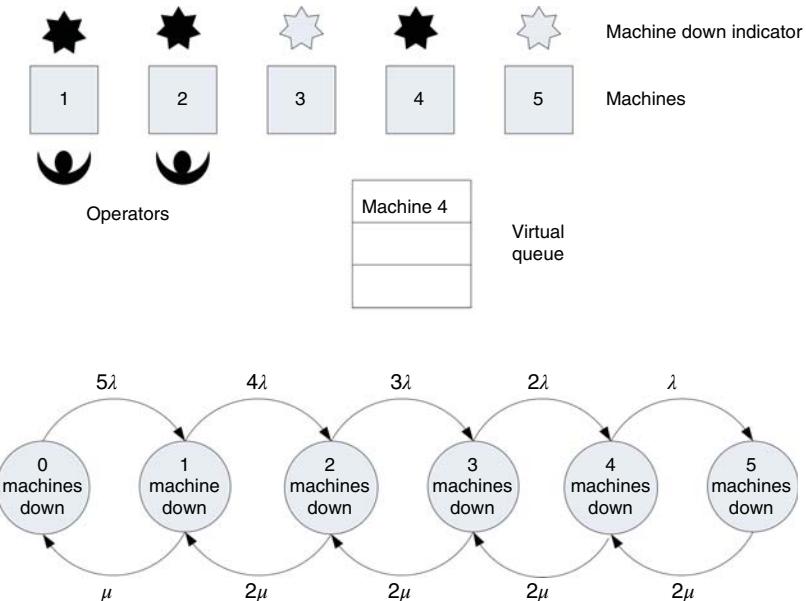


Figure 8.11 Machine interference problem.

each at rate  $\mu$ . Thus, the rate of leaving from state 2 to state 1 is  $2\mu$ . Because there are only two operators in this illustration, the maximum rate is  $2\mu$  for states 2–5.

Notice that the customer in this system is the machine. Even though the machines do not actually move to line up in a queue, they form a virtual queue for the operators to receive their repair. Let us consider an example of this system.

### EXAMPLE 8.6 Machine Interference Model

Suppose a manufacturing system contains five machines, each subject to randomly occurring breakdowns. A machine runs for an amount of time that is an exponential random variable with a mean of 10 hours before breaking down. At present, there are two operators to fix the broken machines. The amount of time that an operator takes to service the machines is exponential with a mean of 4 hours. An operator repairs only one machine at a time. If more machines are broken down than the current number of operators, the machines must wait for the next available operator for repair. They form a first-in, first-out (FIFO) queue to wait for the next available operator. The number of operators required to tend to the machines in order to minimize down time in a cost-effective manner is desired. Assume that it costs the system \$60 per hour for each machine that is broken down. Each operator is paid \$15 per hour regardless of whether they are repairing a machine or not.

*Solution to Example 8.6:* The arrival rate of an *individual machine* is  $\lambda = 1/10$  per hour and the service rate of  $\mu = 1/4$  per hour for each operator. In order to decide the most appropriate number of operators to tend the machines, a service criteria or a way to measure the cost of the system is required. Since costs are given, let us formulate how much a given system configuration costs.

The easiest way to formulate a cost is to consider what a given system configuration costs on a per time basis, for example, the cost per hour. Clearly, the system costs  $15 \times c$  (\$ per hour) to employ  $c$  operators. The problem also states that it costs \$60 per hour for each machine that is broken down. A machine is broken down if it is waiting for an operator or if it is being repaired by an operator. Therefore, a machine is broken down if it is in the queuing system.

In terms of queuing performance measures,  $L$  machines can be expected to be broken down at any time in steady state. Thus, the expected steady-state cost of the broken down machines is  $60 \times L$  per hour. The total expected cost per hour,  $E[TC]$ , of operating a system configuration in steady state is thus

$$E[TC] = 60 \times L + 15 \times c \quad (8.38)$$

Thus, the total expected cost,  $E[TC]$ , can be evaluated for various values of  $c$  and the system that has the lowest cost determined.

Using the *QTSPplus* software, the necessary performance measures can be calculated and tabulated. Within the *QTSPplus* software, you must choose the model category *Multiple Servers* and then choose the *Markov Multi-Server Finite Source Queue without Spares* model. Figure 8.12 illustrates the inputs for the case of one server with five machines. Note that the software requests the time between arrivals,  $1/\lambda$ , and the mean service time,  $1/\mu$ , rather than  $\lambda$  and  $\mu$ .

Table 8.3 shows the results of the analysis. Table 8.3 indicates that as the number of operators increases, the expected cost reaches its minimum at  $c = 2$ . As the number of

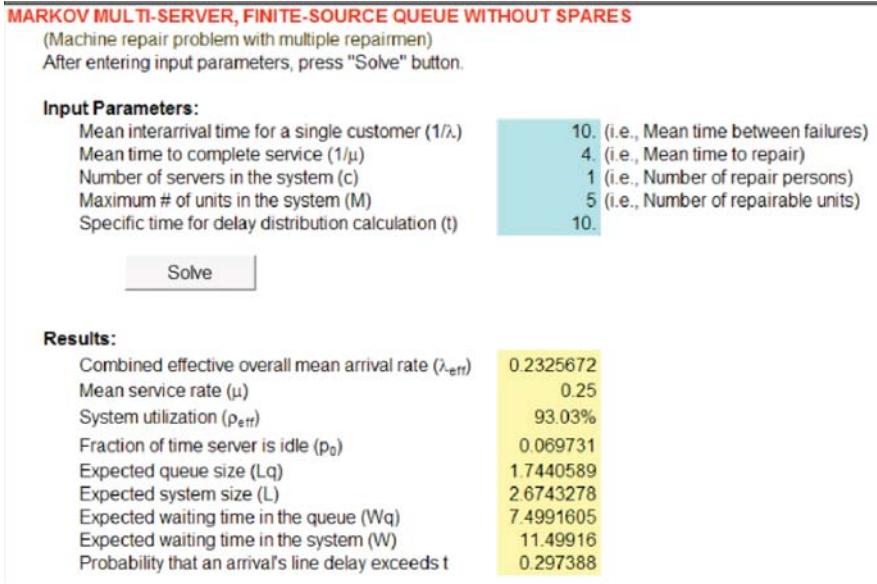


Figure 8.12 Machine interference problem in QTSPPlus.

TABLE 8.3 Tabulated Results for Example 8.6

$\lambda$	0.10	0.10	0.10	0.10	0.10
$\mu$	0.25	0.25	0.25	0.25	0.25
$c$	1	2	3	4	5
$N$	5	5	5	5	5
$K$	5	5	5	5	5
System	M/M/1/5/5	M/M/2/5/5	M/M/3/5/5	M/M/4/5/5	M/M/5/5/5
$L$	2.674	1.661	1.457	1.430	1.429
$L_q$	1.744	0.325	0.040	0.002	0.000
$B$	0.930	1.336	1.417	1.428	1.429
$B/c$	0.930	0.668	0.472	0.357	0.286
$E[TC]$	\$175.460	\$129.655	\$132.419	\$145.816	\$160.714
$\overline{MU}$	0.465	0.668	0.709	0.714	0.714

operators is increased, the machine utilization increases but levels off. The machine utilization is the expected number of machines that are not broken down divided by the number of machines:

$$\text{Machine utilization} = \overline{MU} = \frac{k - L}{k} = 1 - \frac{L}{k} \quad (8.39)$$

## 8.4 NON-MARKOVIAN QUEUES AND APPROXIMATIONS

So far, the queuing models that have been analyzed all assume a Poisson arrival process and exponential service times. For other arrival and service processes, only limited or approximate results are readily available. There are two cases worth mentioning here. The first

**TABLE 8.4 Results M/G/1 and M/D/1**

Model	Parameters	$L_q$
M/G/1	$E[ST] = \frac{1}{\mu}; \text{Var}[ST] = \sigma^2; r = \frac{\lambda}{\mu}$	$L_q = \frac{\lambda^2\sigma^2 + r^2}{2(1-r)}$
M/D/1	$E[ST] = \frac{1}{\mu}; \text{Var}[ST] = 0; r = \frac{\lambda}{\mu}$	$L_q = \frac{r^2}{2(1-r)}$

case is the M/G/1 queue and the second case is an approximation for the GI/G/c queuing system. Recall that G represents any general distribution. In other words, the results will hold regardless of the distribution. Also, GI refers to an arrival process which has the time between arrivals as IID random variables with any distribution.

For the M/G/1 model with a service distribution having a mean  $E[ST] = 1/\mu$  and variance  $\sigma^2$ , the expected number in the system is

$$L_s = \frac{\lambda^2\sigma^2 + r^2}{2(1-r)} + r \quad (8.40)$$

From this formula for the expected number in the system, the other performance measures can be obtained via Little's formula. Notice that only the mean and the variance of the service time distribution are necessary in this case.

For the case of the GI/G/c queue, a number of approximations have been influenced by an approximation for the GI/G/1 queue that first appeared in Kingman [1964]. His single-server approximation is shown below:

$$W_q(\text{GI/G/1}) \approx \left( \frac{c_a^2 + c_s^2}{2} \right) W_q(\text{M/M/1}) \quad (8.41)$$

In this equation,  $W_q(\text{M/M/1})$  denotes the expected waiting time in the queue for the M/M/1 model  $c_a^2$  and  $c_s^2$  and represents the squared coefficient of variation for the interarrival time and service time distributions. Recall that for a random variable,  $X$ , the squared coefficient of variation is given by  $c_X^2 = \text{Var}[X]/(\text{E}[X])^2$ . Whitt [1983] used a very similar approximation for the GI/G/c queue to compute the traffic congestion at each node in a queuing network for his Queuing Network Analyzer:

$$W_q(\text{GI/G/c}) \approx \left( \frac{c_a^2 + c_s^2}{2} \right) W_q(\text{M/M/c}) \quad (8.42)$$

A discussion of queuing approximations of this form as well as additional references can be found in Whitt [1993].

Thus, to approximate the performance of a GI/G/c queue, you need only the first two moments of the interarrival and service time distributions and a way to compute the waiting time in the queue for a M/M/c queuing system. These results are useful when trying to verify and validate a simulation model of a queuing system, especially in the case of a system that consists of more than one queuing system organized into a network. Before examining that more complicated case, some of the issues related to using Arena™ to simulate single-queue systems should be examined.

## 8.5 SIMULATING SINGLE QUEUES IN ARENA

In this section, we illustrate how to model queuing situations in Arena<sup>TM</sup>. We begin by illustrating how to model the machine interference system. The unique feature of this model is that it represents a closed queuing network. In other words, the entities within the model are created but never disposed. Then, we examine a health-care-oriented example, which will illustrate how a simple change in the queue processing necessitates using simulation.

### 8.5.1 Machine Interference Optimization Model

This section presents a simulation model to analyze the M/M/c/k/k machine interference model. Naturally, the queuing formulas of the previous section could be used to calculate the performance of this system; however, with only minor changes to the system characteristics, the formulas cannot be applied. *OptQuest* will also be introduced within the Arena<sup>TM</sup> environment. *OptQuest* is an add-on to Arena<sup>TM</sup> which provides heuristic-based simulation optimization capabilities.

To develop this model, the standard modeling recipe can be used:

- What is the system? What information is known by the system?
- What are the required performance measures?
- What are the entities? What information must be recorded or remembered for each entity?
- What are the resources that are used by the entities? Which entities use which resources and how?
- What are the process flows? Sketch the process or make an activity flow diagram
- Develop pseudo-code for the situation.
- Implement the model in Arena<sup>TM</sup>.

The system consists of the machines and the operators. The system must know about the number of operators present, the number of machines, how the machines breakdown (the running time to breakdown), the repair time, the costs associated with the broken down machines, and the cost per hour for the operators. Thus, the costs can be modeled as variables, the running time as a distribution (expression), and the repair time as a distribution (expression) within Arena<sup>TM</sup>. As described in Example 8.6, the expected total cost per hour of operating the system needs to be estimated. This involves estimating the average number of machines that are broken down. In addition, to be consistent with Example 8.6, the utilization of the machines needs to be estimated.

Now, the potential entities and resources within this system can be discussed. The operators clearly act as resources within the system. The most natural entity is the machine, since as in Example 8.6, the machines must wait in queue for the operators to perform the repair. However, the utilization of the machines is also required. Are the machines resources or entities? In this situation, it is useful to think a little harder about what is actually flowing in the system. When a machine stops running, a repair job is initiated for an operator. It is a repair job that waits on a list for the next available operator. When the repair job is completed, the machine (associated with the repair job) is put back into service and essentially works on a production job. Thus, the modeling of the entity as a job might be useful from a conceptual standpoint.

**Exhibit 8.1** Pseudo-Code for Machine Interference Model

---

```

CREATE 5 jobs
ASSIGN A:
    myType = 1, 1 indicates production job
    vNumInProd = vNumInProd +1
END ASSIGN
PROCESS Production
    SEIZE 1 unit of machine
    DELAY for running time = EXPO(10) hours
    RELEASE 1 unit of machine
END PROCESS
ASSIGN
    myType = 2, 2 indicates repair job
    vNumInProd = vNumInProd -1
    vNumInRepair = vNumInRepair +1
END ASSIGN
PROCESS Repair
    SEIZE 1 unit of Operator
    DELAY for repair time = EXPO(4) hours
    RELEASE 1 unit of Operator
END PROCESS
ASSIGN vNumInRepair = vNumInRepair -1
GOTO A:

```

---

A production job requires a machine to run. A repair job requires the operator for repair. What is the maximum number of repair jobs that can be in the system? There can never be more than  $k$  (#machines) broken down. Thus, there can never be more than  $k$  repair jobs. There can never be more than  $k$  (#machines) running. Thus, there can never be more than  $k$  production jobs. Because a machine is either broken down or running, it should be clear that at any time  $t$ ,

- $\text{Number of machines} = \text{Number of broken machines} + \text{Number of running machines}$
- $\text{Number of machines} = \text{Number of repair jobs} + \text{Number of production jobs}$

If the jobs are modeled as entities, then the machines can be modeled as a resource (seized by production jobs). This will allow Arena<sup>TM</sup> to calculate the utilization of the machines automatically via its resource statistics.

Now, let us describe the process flow associated with the jobs. If a job can be thought of as simply changing type (between repair and production), then an activity cycle diagram can be developed for the life of a job. An activity cycle diagram is a special type of activity flow diagram in which the life of the entity *cycles*. In Figure 8.13, this is illustrated by the loop. Notice that in this case, there is no concept of create and dispose.

The Arena-like pseudo-code is given in Exhibit 8.1. In the pseudo-code, two variables have been used to keep track of the number of jobs in production and the number of jobs in repair. First the five jobs are created. Then the type of the job is set as production and the job goes through the production process (SEIZE, DELAY, RELEASE). When the job is done with production, the machine needs tending. The job is changed to a repair job and the job goes to the repair process. After the repair job is completed, it goes back to production. Now, these conceptual models must be translated into something more suitable to Arena<sup>TM</sup>.

The building of the Arena<sup>TM</sup> model follows closely the pseudo-code. The completed model can be found in the file *Ch8-MMckk-Example.doe* that accompanies this chapter.

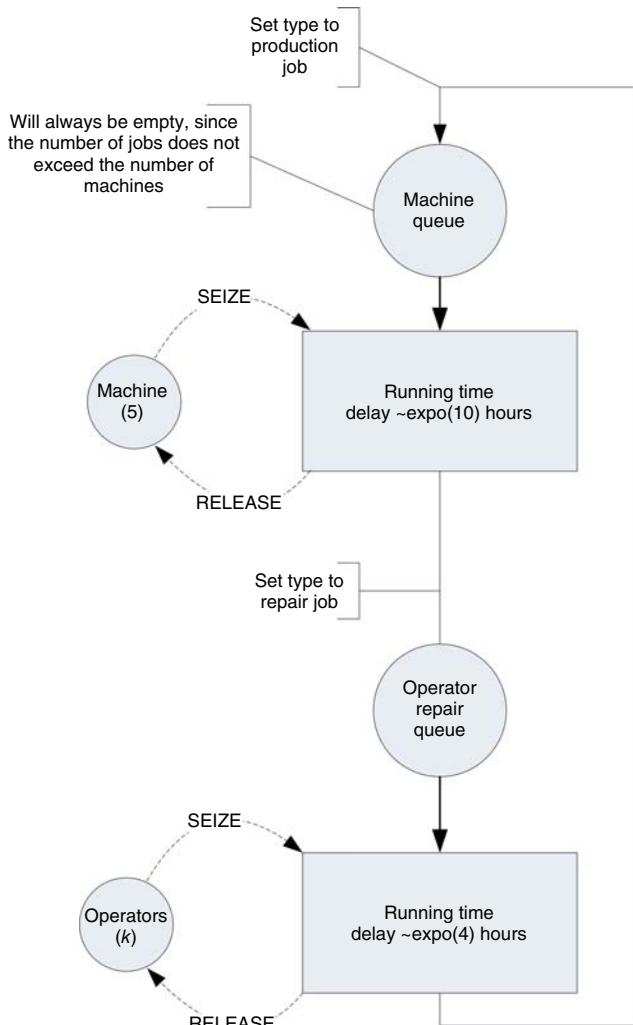


Figure 8.13 Activity cycle diagram for a job.

You should first define the entities, variables, expressions, and resources in the model as illustrated in Figures 8.14 and 8.15. Next, lay down the model's flow modules as illustrated in Figure 8.16. The model follows closely the already described activity cycle diagram and pseudo-code.

Figure 8.17 indicates that the CREATE module depends on the number of machines defined. There will be  $MR(Machines)$  jobs created at time 0.0 and then the CREATE module will stop its creation.  $MR(resource\ name)$  is a special-purpose function in Arena<sup>TM</sup> that returns the current number of units scheduled for a resource.  $MR$  represents the capacity of the resource at any time  $t$ . In this case, the capacity is five for the five machines.  $MR$  jobs will be created because that is the most that can be running at any time. Five jobs for five machines. This logic assumes that the system starts with all the machines working on production jobs at the start of the simulation.

Resource - Basic Process			
	Name	Type	Capacity
1	Machines	Fixed Capacity	5
2	Operators	Fixed Capacity	1

Entity - Basic Process		
	Entity Type	Initial Picture
1	ProdJob	Picture.Green Ball
2	RepairJob	Picture.Red Ball

Figure 8.14 Entity and Resource Data Modules

Variable - Basic Process							
	Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics
1	RepairCost			Real	System	1 rows <input type="text" value="60"/>	<input type="checkbox"/>
2	RPCost			Real	System	1 rows <input type="text" value="15"/>	<input type="checkbox"/>
3	vNumInProd			Real	System	0 rows <input type="checkbox"/>	<input type="checkbox"/>
4	vNumInRepair			Real	System	0 rows <input type="checkbox"/>	<input type="checkbox"/>

Expression - Advanced Process				
	Name	Rows	Columns	Expression Values
1	eRunningTime			1 rows <input type="text" value="EXPO(10)"/>
2	eRepairTime			1 rows <input type="text" value="EXPO(4)"/>

Figure 8.15 Variable and expression modules.

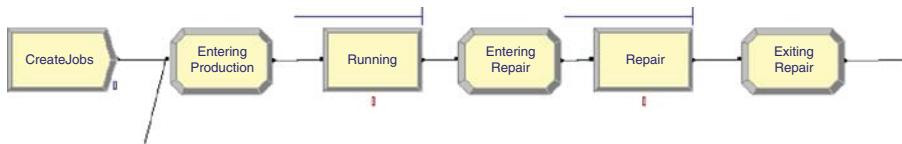


Figure 8.16 Machine interference flowchart modules.

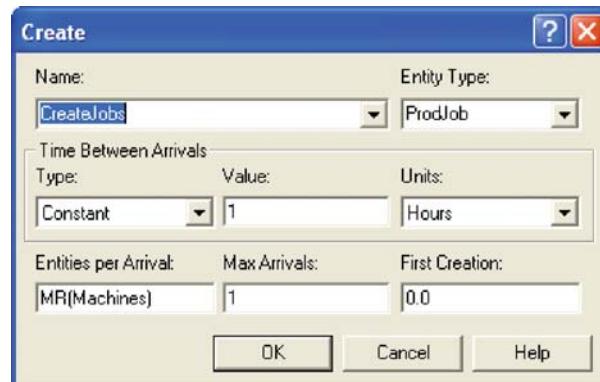
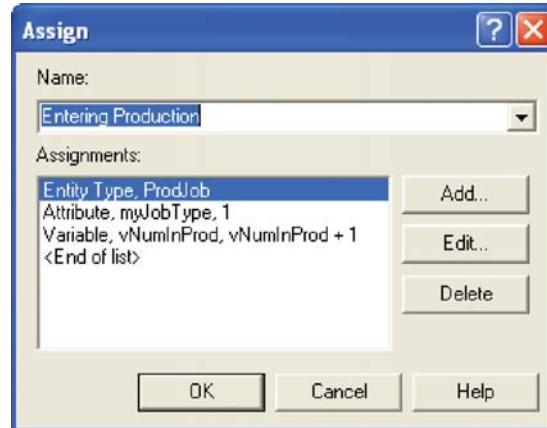
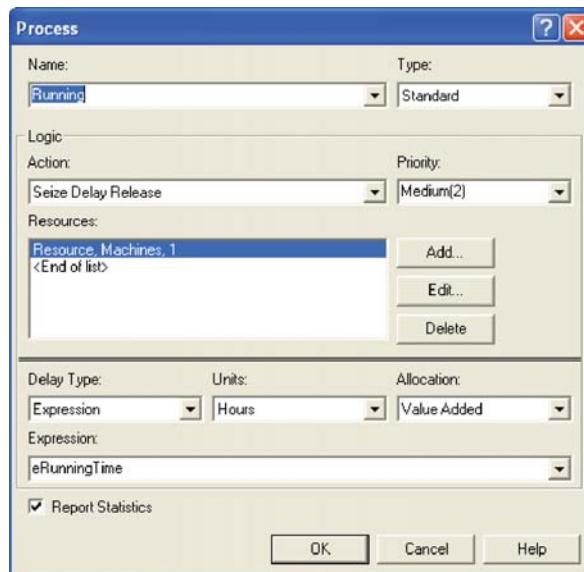


Figure 8.17 CREATE module for machine interference model.



**Figure 8.18** ASSIGN module for machine interference model.



**Figure 8.19** Machine running time PROCESS module.

As the production job enters production on the machines, the number of jobs in production is incremented. A PROCESS module is then used to implement the SEIZE, DELAY, RELEASE logic for the running time of the machine as shown in Figure 8.19. After the machine completes the running time, the production job is changed to a repair job and the number of repair jobs in the system is incremented as shown in Figure 8.20. Finally, the repair job is processed and the number of repair jobs in the system is decremented as shown in Figures 8.21 and 8.22.

In order to collect the total cost on an hourly basis, an OUTPUT statistic should be defined as shown in Figure 8.23. A variable *RPCost* has been defined to represent the cost per hour for the operator (e.g., \$15 per hour). In addition, the repair cost is represented

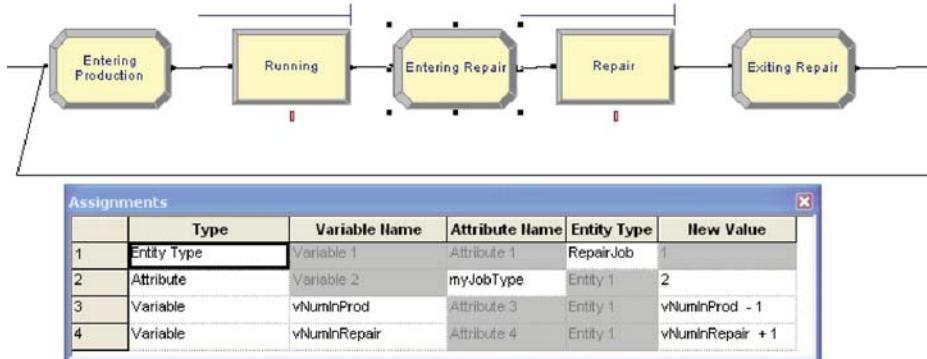


Figure 8.20 Entering repair ASSIGN module.

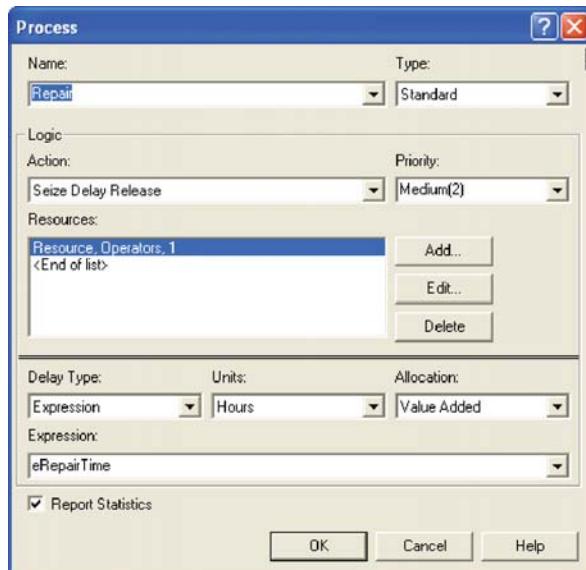


Figure 8.21 Repair PROCESS module.

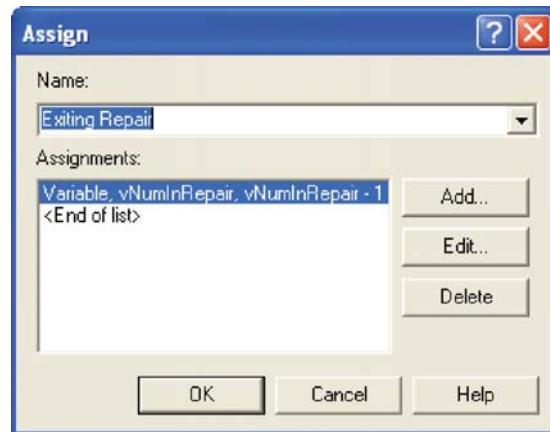
with a variable, *RepairCost*, with an initial value of \$60 per broken machine). The Statistic, *HourlyCostStat*, thus represents equation:  $E[TC] = 15 \times c + 60 \times L$ .

Figure 8.24 illustrates the results of the case of one operator and five machines. The statistics match closely the results from the queuing analysis. The results are based on 21 replications of length 12,000 hours with a warm-up period of 2000 hours.

### 8.5.2 Using OptQuest<sup>4</sup> on the Machine Interference Model

As we saw in the last section, the expected total cost for this system can be easily evaluated by using Arena<sup>TM</sup>. In addition, by utilizing *OptQuest* for Arena<sup>TM</sup>, we can optimize the system to find the number of operators with the lowest cost.

<sup>4</sup>The use of OptQuest requires the professional license of Arena<sup>TM</sup>.



**Figure 8.22** ASSIGN for exiting repair.

Statistic - Advanced Process					
	Name	Type	Expression	Report Label	Output File
1	HourlyCostStat	Output	RPCost*MR(Operators) + RepairCost*(DAVG(InRepairStat))	HourlyCostStat	
2	InProductionStat	Time-Persistent	vNumInProd	InProductionStat	
3	InRepairStat	Time-Persistent	vNumInRepair	InRepairStat	

Double-click here to add a new row.

**Figure 8.23** Defining the total hourly cost.

Time Persistent		
	Average	Half Width
InProductionStat	2.3250	0.00
InRepairStat	2.6750	0.00
<b>Output</b>		
	Average	Half Width
HourlyCostStat	175.50	0.17
Instantaneous Utilization		
	Average	Half Width
Machines	0.4650	0.00
Operators	0.9302	0.00
Number Busy		
	Average	Half Width
Machines	2.3250	0.00
Operators	0.9302	0.00

**Figure 8.24** Arena™ results for one operator and five machines.

If you are following along with the model building process, make sure that your model is open, and go *Tools > OptQuest for Arena™*. *OptQuest* will open up and you should see a dialog asking you whether you want to browse for an already formulated optimization run or to start a new optimization run. Select the start new optimization run button. *OptQuest* will then read your model and start a new optimization model (Figure 8.25), which may take a few seconds, depending on the size of your model. *OptQuest* is similar to the Process Analyzer in some respects. It allows you to define controls and responses for your model. Then you can develop an optimization function that will be used to evaluate the system under the various controls. In addition, you can define a set of constraints that must not be violated by the final recommended solution to the optimization model. It is beyond the scope of this example to fully describe simulation optimization and all the intricacies of this technology. The interested reader should refer to April et al. [2001] and Glover et al. [1999] for more information on this technology. This example will simply illustrate the possibilities of using this technology. A tutorial on the use of *OptQuest* is available in the *OptQuest* help files.

Using the controls, double click on the resource Operators and change the lower bound and the upper bound on the range for the control to 1 and 10, respectively. This defines the range of values that *OptQuest* will search over to find a potential optimal solution. The controls act as the decision variables within the optimization model. Then, in the responses area, select the *HourlyCostStat* as a response to include in the model. Finally, you should set up an objective function.

Within *OptQuest*, you can build an objective function by using the controls and responses that were included in the optimization model. The objective function is quite simple here. The *HourlyCostStat* response should be selected as the objective function as shown in Figure 8.26. More general objective functions can be easily formed using the objective function editing capabilities of *OptQuest*. The objective function will be used by *OptQuest* to direct the search.

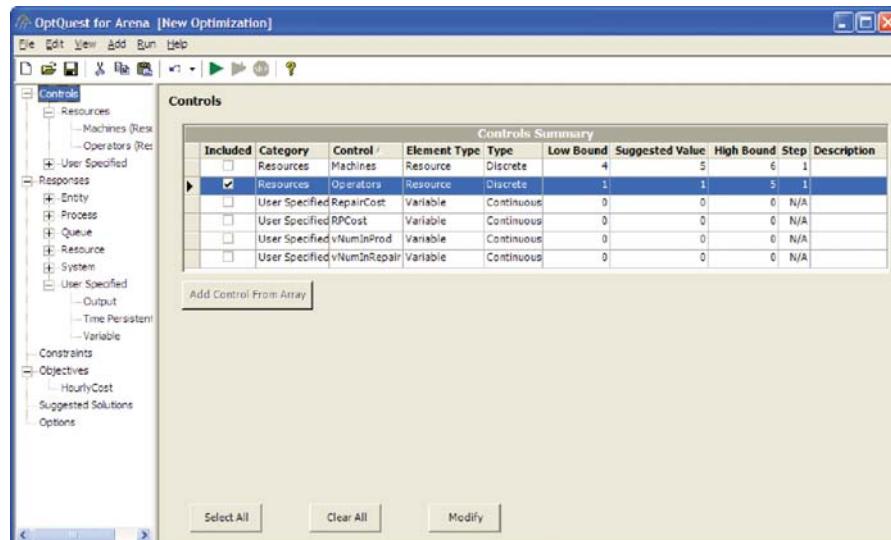
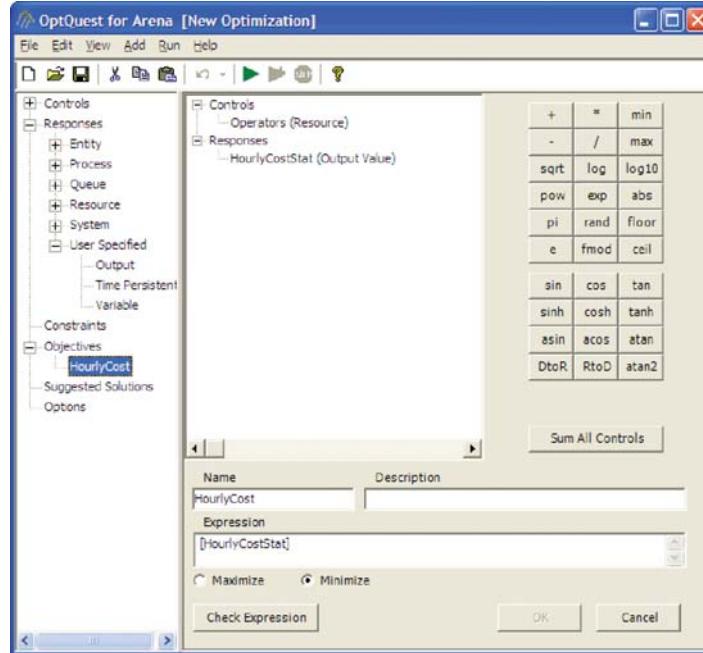


Figure 8.25 OptQuest for Arena™.



**Figure 8.26** Setting up the objective function in OptQuest.

*OptQuest* will use its intelligent search technology to run multiple replications at each combination of the decision variables. *OptQuest* will then evaluate the objective function and decide on new values for the decision variables in order to minimize (or maximize) its value. *OptQuest* uses a number of meta-heuristics (primarily based on Tabu Search, see Glover and Laguna [1997]) to direct the search.

The *OptQuest* optimization options tab allows you to specify control parameters for how *OptQuest* controls the number of replications (Figure 8.27). The default settings for this simple search will be used. Press the triangle like run button and execute the search. You should see a figure that tabulates the progress of the search at each simulation, see Figure 8.28. In the figure, *OptQuest* has recommended a solution having two operators, which, based on the analytical results, is the optimal solution. Once you find a set of possibly optimal solutions, you can use either *OptQuest*'s refine solution tab or the Process Analyzer to statistically select the best system. This example is too limited to fully explore the capabilities of *OptQuest*. The use of *OptQuest* will be examined again on a more complicated problem in Chapter 11.

In the next section, we explore a slightly more complicated situation that does not lend itself to analytical analysis via queuing formulas.

### 8.5.3 Modeling Balking and Reneging

Example 8.7 illustrates the modeling of a queuing situation for which an analytical solution is not readily available. This situation has multiple types of customers with different priorities, different service times, and, in the case of one type of customer, the desire (or ability) to renege from the system.

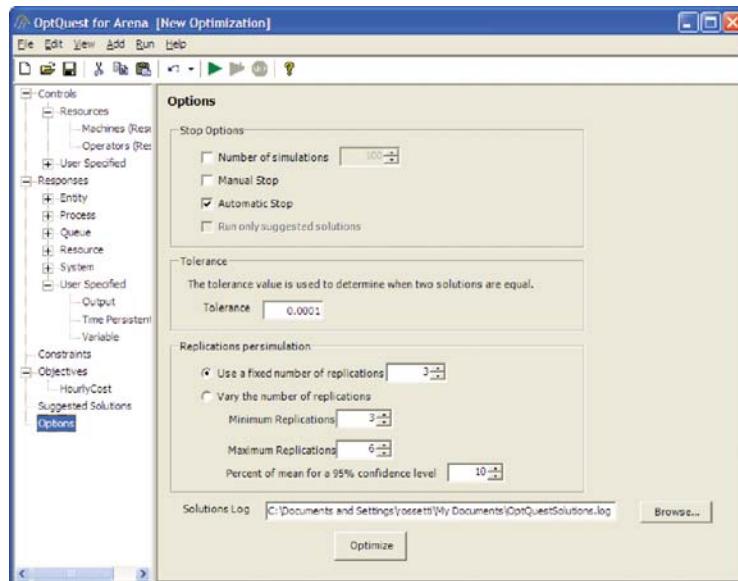


Figure 8.27 OptQuest optimization options.

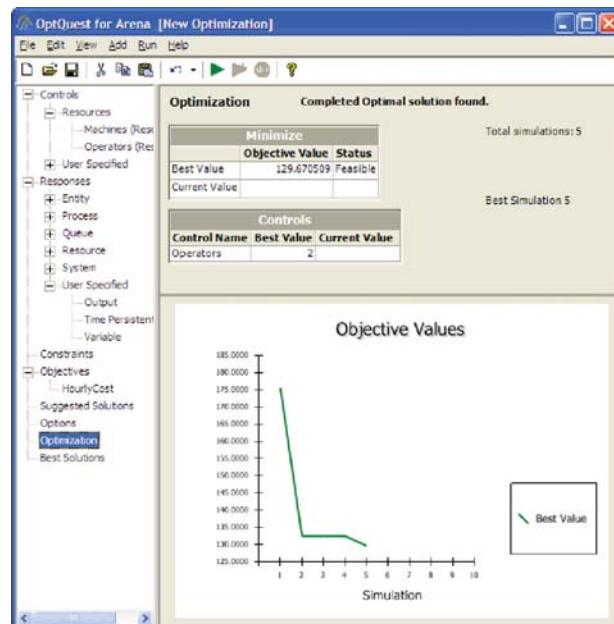


Figure 8.28 OptQuest search results.

### EXAMPLE 8.7 Walk-in Health Care Clinic

A walk-in health care clinic has analyzed their operations and they have found that they can classify their walk-in patients into three categories: high priority (urgent need of medical attention), medium priority (need standard medical attention), and low priority (nonurgent need of medical attention). On a typical day during the period of interest, there are about 15 arrivals per hour with 25% being high priority, 60% being medium priority, and the remaining being low priority. The clinic is interested in understanding the waiting time for patients at the clinic. Upon arrival to the clinic, the patients are triaged by a nurse into one of the three types of patients. This takes only 2–3 minutes uniformly distributed. Then, the patients wait in the waiting room based on their priority. Patients with higher priority are placed at the front of the line. Patients with the same priority are ordered based on a FCFS basis. The service time distributions of the customers are given as follows.

Priority	Service Time Distribution (in Minutes)
High	Lognormal(38, 8)
Medium	Triangular(16, 22, 28)
Low	Lognormal(12, 2)

The clinic has four doctors on staff to attend to the patients during the period of interest. They have found through a survey that if there are more than 10 people waiting for service, an arriving low priority patient will exit before being triaged. Finally, they have found that the nonurgent (low priority) patients may depart if they have to wait longer than  $15 \pm 5$  minutes after triage. That is, a nonurgent patient may enter the clinic and begin waiting for a doctor, but if they have to wait more than  $15 \pm 5$  minutes (uniformly distributed), they will decide to renege and leave the clinic without getting service. The clinic would like to estimate the following:

- (a) the average system time of each type of patient,
- (b) the probability that low priority patients balk,
- (c) the probability that low priority patients renege,
- (d) the distribution of the number of customers waiting in the doctor queue.

*Solution to Example 8.7:* For this problem, the system is the walk-in clinic, which includes doctors and a triage nurse who serve three types of patients. The system must know how the patients arrive (time between arrival distribution), how to determine the type of the patient, the triage time, the service time by type of patient, and the amount of time that a low priority patient is willing to wait.

To determine the type of the patient, a discrete distribution (DISC) can be used. The service time distributions depend on the patient type and can be easily held in an arrayed EXPRESSION. In fact, this information can be held in expressions as illustrated in Figure 8.29. In the DISC() distribution, 1 equals high priority, 2 equals medium priority, and 3 equals low priority. In addition, the system must know the balk criteria for the low priority patients. This information is modeled with variables as illustrated in Figure 8.30.

The entity for this model is the patient. Patients are created according to a particular type, enter the system, and then depart. Thus, a key attribute for patients should be their type. In

Expression - Advanced Process				
	Name	Rows	Columns	Expression Values
1	PatientServiceTime	3		3 rows
2	PatientTypeDist			1 rows
3	RenegTimeDist			1 rows

Expression Values	
1	LOGN(38,8)
2	TRIA(16,22,28)
3	LOGN(12,2)

Expression Values	
	DISC(0.25,1,0.85,2,1,0,3)

Expression Values	
	UNIF(10,20)

Figure 8.29 Expressions for walk-in clinic model.

Variable - Basic Process							
	Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics
1	vBalkCriteria			Real	System	1 rows	10
2	vSearchNumb			Real	System	0 rows	<input type="checkbox"/>

Figure 8.30 Walk-in clinic variable module.

addition, the required performance measures require that the arrival time of the patient be stored so that the total time in the system can later be calculated. There are two resources for this system: triage nurse and doctors, with 1 and 4 units, respectively.

The process flow for this system is straightforward, except for the reneging of low priority patients as illustrated in Exhibit 8.2. In the pseudo-code, the patient is created and then the type is assigned. If the type of patient is of low priority and the doctor's queue has become too large, then the low priority patient balks. Otherwise, the patient is triaged by the nurse. After triage, the patient will wait in the doctor's queue until their renege time is up or they get served. After the doctor serves the patient, statistics are recorded and the patient departs the system.

Modeling the reneging of the patients within Arena™ requires additional effort. There are no magic dialog boxes or modules that directly handle reneging. The key to modeling the reneging described in this problem within Arena™ is to remember how Arena™ processes entities. In particular, from your study in Chapter 4, you saw that no model logic is executed unless an entity is moving through the model. In addition, you know that an entity that is in a queue is no longer moving. Thus, an entity within a queue cannot remove itself from the queue! While this presents a difficulty, it also indicates the solution to the problem.

If the entity that represents the low priority patient cannot remove itself from the queue, then *some other* entity must do the removal. The only other entities in the system are other patients and it seems both unrealistic and illogical to have another patient remove the reneging patient. Thus, you can only conclude that another (logical) entity needs to be created to somehow implement the removal of the reneging patient.

As you have learned, there are two basic ways to create entities: the CREATE module and the SEPARATE module. Since there is no clear pattern associated with when the patient will renege, this leaves the SEPARATE module as the prime candidate for implementing reneging. Recall that the SEPARATE module works by essentially cloning the entity that goes through it in order to create the specified number of clones. Now the answer should be

**Exhibit 8.2** Pseudo-Code for Walk-In Clinic Model

---

```

CREATE patients
ASSIGN myType = PatientTypeDist
DECIDE
    IF (myType == 3) and (NQ(Doctor's Q) ≥ vBalkCriteria) THEN
        RECORD balk
        DISPOSE patient
    ELSE
        SEIZE 1 unit of triage nurse
        DELAY for UNIF(2,3) minutes
        RELEASE 1 unit of triage nurse
        Handle reneging patients
        SEIZE 1 unit of Doctors
        DELAY for service based on patient type
        RELEASE 1 unit of Doctors
        RECORD patient's system time
        DISPOSE
    END IF
END DECIDE

```

---

clear. The low priority patient should clone himself prior to entering the queue. The clone will be responsible for removing the cloned patient from the queue if the delay for service is too long.

Figure 8.31 illustrates the basic ideas for modeling reneging in the form of an activity diagram. Using a SEPARATE module, the patient entity can be duplicated prior to the entity entering the doctor's queue. Now the two flows can be conceptualized as acting in parallel. The (original) patient enters the doctor queue. If it has to wait, it stops being the active entity, stops moving, and gives control back the event scheduler. At this point, the duplicate (clone) entity begins to move.

Within the simulation, no actual simulation time has advanced. The clone then enters the delay module that represents the time until reneging. This schedules an event to represent this delay and the clone's progress stops. The event scheduler then allows other entities to proceed through their process flows. One of those other entities might be the original patient entity. If the original entity continues its movement and gets out of the queue before the clone finishes its delay, then it will automatically be removed from the queue by the doctor resource and be processed. If the clone's delay completes before the original exits the queue, then when the clone becomes the active entity, it will remove the original from the queue and proceed to being disposed. If the clone does not find the original in the queue, then the original must have proceeded and the clone can simply be disposed. One can think of this as the patient, setting an alarm clock (the event scheduled by the duplicate) for when to renege. To implement these ideas within Arena™, you will need to use the SEPARATE, SEARCH, and REMOVE modules. Now, let us take a look at the entire model within Arena™.

Figure 8.32 provides an overview of the walk-in clinic model that follows closely the previously discussed pseudo-code. The CREATE module has a time between arrival specified as Random(expo) with a mean of 6 minutes. The next ASSIGN module simply assigns the arrival time and the type of patient and then changes the Entity Type and Entity Picture based on some sets that have been defined for each type of patient. This is shown in Figure 8.33.

The Triage and Doctor PROCESS modules are done similarly to how you have implemented many of the prior models. In order to implement the priority for the patients by

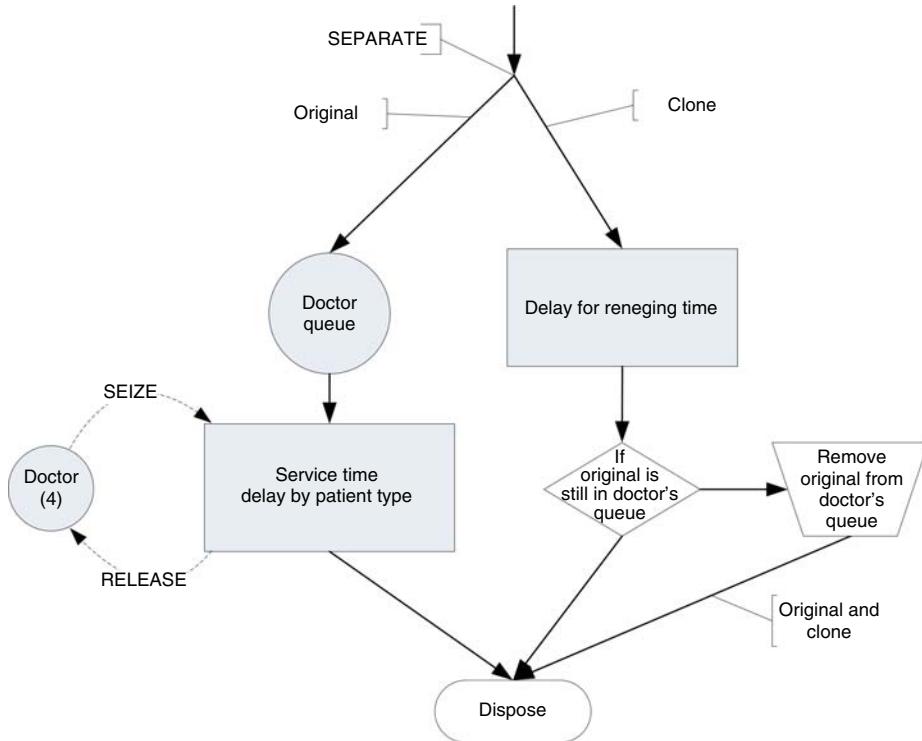


Figure 8.31 Activity diagram for modeling reneging.

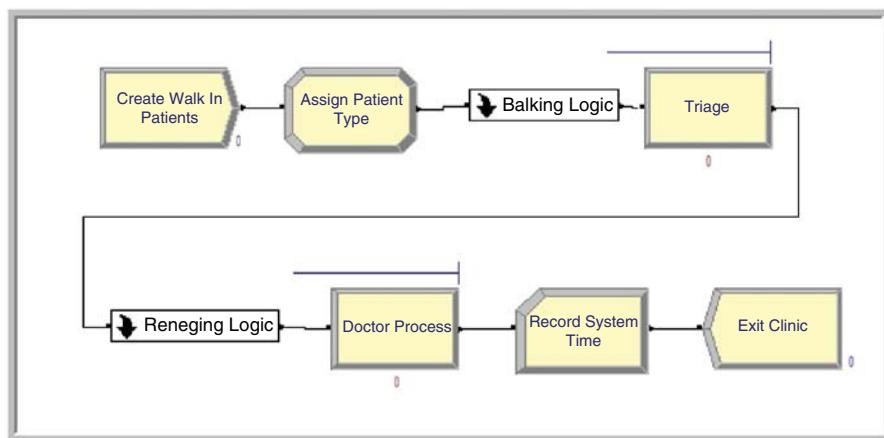


Figure 8.32 Overview of walk-in clinic model.

type, the QUEUE module can be used. In this case, the *myType* attribute can be used with the lowest attribute value as shown in Figure 8.34. You should attempt to build this model or examine the supplied Arena™ file for all the details.

The balking and reneging logic have been placed inside two different submodels. Balking only occurs for nonurgent patients, so the type of patient is first checked. If the

Assignments				
	Type	Attribute Name	Other	New Value
1	Attribute	myArrivalTime	J	TNOW
2	Attribute	myType	J	PatientTypeDist
3	Other	Attribute 3	Entity.Type	MEMBER(PatientTypeSet,myType)
4	Other	Attribute 4	Entity.Picture	MEMBER(EntityPictureSet,myType)

Double-click here to add a new row.

Figure 8.33 Assigning the type of patient.

Queue - Basic Process					
	Name	Type	Attribute Name	Shared	Report Statistics
1	Triage.Queue	First In First Out	Attribute 1	<input type="checkbox"/>	<input checked="" type="checkbox"/>
2	DoctorProcess.Queue	Lowest Attribute Value	myType	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Double-click here to add a new row.

Figure 8.34 Using the QUEUE module to rank the queue.

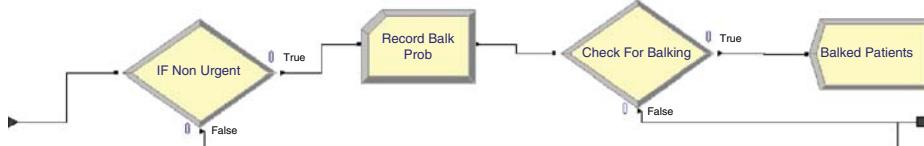


Figure 8.35 Balking logic submodel.

patient is of type nonurgent, whether a balk will occur can be recorded with the expression,  $NQ(DoctorProcess.Queue) \geq vBalkCriteria$ . This expression evaluates to 1 for true or 0 for false. Thus, the probability of balking can be estimated. The patients who actually balk are then disposed (Figure 8.35).

The reneging logic is more complicated. Figure 8.36 presents an overview of the submodel logic for reneging.

In the logic, the patient's type is checked to see if the patient is of type nonurgent. If so, the entity is sent to a SEPARATE module. The original entity simply exits the submodel (to proceed to the queue for the doctor resource). The duplicate then enters the DELAY module that schedules the time until reneging. After exiting the reneging delay, an ASSIGN module is used to set a variable ( $vSearchNumber$ ) equal to  $Entity.SerialNumber$ . Recall from Chapter 5 that  $Entity.SerialNumber$  is a unique number given to each entity when created. When the original entity was duplicated by the SEPARATE module, the duplicate also has this same number assigned. Thus, you can use this number to have the clone search for the original entity within the queue. The SEARCH module allows the searching of a batch, queue, or expression for a particular expression.

As illustrated in Figure 8.37, the  $DoctorProcess.Queue$  is searched starting at the first entity in the queue (rank 1) to last entity in queue(the entity at rank  $NQ$ ). The search proceeds to find the first entity where  $vSearchNumber == Entity.SerialNumber$ . As the SEARCH module indicates, if the search condition is true, the global variable,  $J$ , is set to

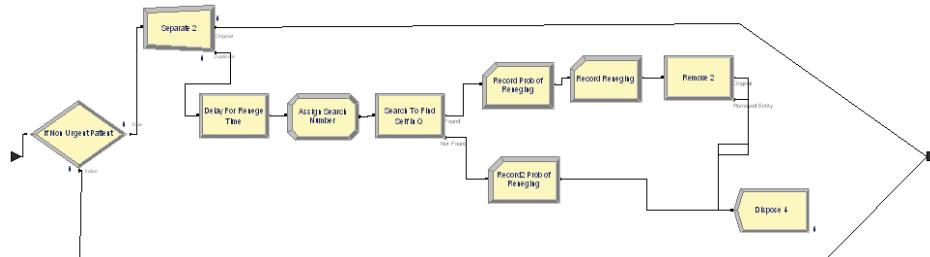


Figure 8.36 Reneging logic submodel.

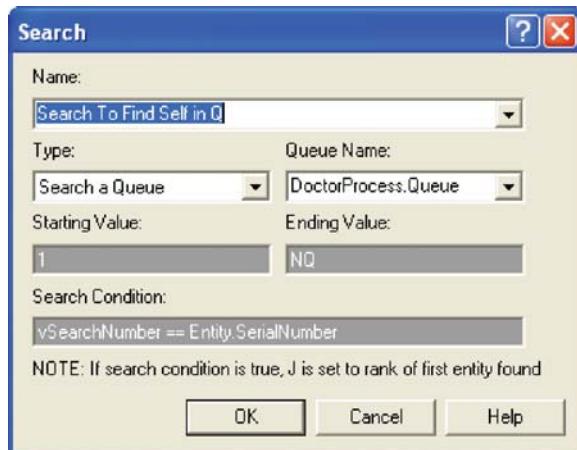


Figure 8.37 SEARCH module for renege logic.

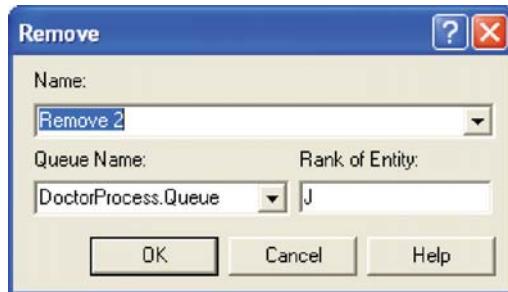
the rank of the first entity satisfying the condition. Thus, after the SEARCH module completes, the variable,  $J$ , can be used to see if an entity was found. An entity will have been found if  $J > 0$  and not found if  $J = 0$ . The SEARCH module has two exit points: one for if the search found something, the other for the case of not finding something. If no entity was found, then the duplicate can simply be disposed because the original is no longer in the queue. If an entity was found then the variable,  $J$ , can be used within the REMOVE module to remove the appropriate entity from the queue. The two RECORD modules on both paths after the SEARCH modules use the fact that the Boolean expression,  $J > 0$ , will indicate whether or not there was a renege. This is can be observed as an expression in the RECORD modules to collect the probability of reneging as illustrated in Figure 8.38.

If  $J > 0$ , then the entity at rank  $J$  can be removed from the *DoctorProcess.Queue* as illustrated in Figure 8.39. The rest of the model is relatively straightforward and you are encouraged to explore the final dialog boxes.

Assuming that the clinic opens at 8 a.m. and closes at 6 p.m., the simulation was set up for 30 replications to yield the results shown in Figure 8.40. It appears that there is a relatively high chance (about 29%) that a nonurgent patient will renege. This may or may not be acceptable in light of the other performance measures for the system. The reader is asked to further explore this model in the exercises, including the implementation to collect statistics on the number of customers waiting in the doctor queue.



**Figure 8.38** Recording whether reneging occurred.



**Figure 8.39** REMOVE module for removing reneging entity.

Tally		
Expression	Average	Half Width
Record Balk Prob	0.00222222	0.00
Record Prob of Reneging	0.2884	0.07

**Figure 8.40** Example output for walk-in clinic.

While some analytical work has been done for queuing systems involving balking and reneging, simulation allows for the modeling of more realistic types of queuing situations as well as even more complicated systems. In the next section, we introduce a very useful construct that enables condition-based signaling and control of entity movement. When the entity waits for the condition, it waits in a queue. This permits a wider variety of modeling involving queues.

## 8.6 HOLDING AND SIGNALING ENTITIES

In this section, we explore the use of two new modules: HOLD and SIGNAL. The HOLD module allows entities to be held in a queue based on three different options: (i) wait for signal, (ii) infinite hold, and (iii) scan for condition. The SIGNAL module broadcasts a signal to any HOLD modules that are listening for the specified signal.

To understand the usage of the HOLD module, we must better understand the three possible options:

1. *Wait for Signal.* The wait for signal option holds the entities in a queue until a specific numerical value is broadcast via a SIGNAL module. The numerical value is specified via the *Wait for Value* field of the module. This field can be a constant, an expression (which evaluates to a number), or even an attribute of an entity. In the case of an attribute, each incoming entity can wait for a different numerical value. Thus, when entities waiting for a specific number are signaled with that number, they exit the HOLD module. The other entities stay in the queue.
2. *Infinite Hold.* The infinite hold option causes the entities to wait in the queue until they are physically removed (via a REMOVE module).
3. *Scan for Condition.* The scan for condition option holds the entities in the queue until a specific condition becomes true within the model. Every time an event occurs within the model, *all* of the scan for conditions are evaluated. If the condition is true, the waiting entities are permitted to leave the module immediately (at the current time). Since moving entities may cause conditions to change, the check of the condition is performed until no entities can move at the current time. As discussed in Chapter 4, this essentially allows entities to move from the waiting state into the ready state. Then the next event is executed. Because of the condition checking overhead, models that rely on scan for condition logic may execute more slowly.

In the wait for signal option, the entities wait in a single queue, even though the entities may be waiting on different signal values. The entities are held in the queue according to the queue discipline specified in the QUEUE module. In the state diagram of Chapter 4, the entities are in the condition delayed state, waiting on the condition associated with a particular signal. The default discipline of the queue is FIFO. There can be many different HOLD modules that have entities listening for the same signal within a model. The SIGNAL is broadcast globally and any entities waiting on the signal are released and moved into the ready state. The order of release is essentially arbitrary if there is more than one HOLD module with entities waiting on the signal. If you need a precise ordering of release, then send all entities to another queue that has the desired ordering (and then release them again), or use a HOLD/REMOVE combination instead.

The infinite hold option places the entities in the dormant state discussed in Chapter 4. The entities will remain in the queue of the HOLD module until they are removed by the active entity via a REMOVE or PICKUP module. An example of this is presented in Chapter 10.

The scan for condition option is similar in some respects to the wait for signal option. In the scan for condition option, the entities are held in the queue (in the condition delayed state) until a specific condition becomes true. However, rather than another entity being required to signal the waiting entities, the simulation engine notifies waiting entities if the condition become true. This eases the burden on the modeler but causes more computational time during the simulation. In almost all instances, a HOLD with a scan for condition option can be implemented with a HOLD and a wait for signal option by carefully thinking about when the desired condition will change and using an entity to send a signal at that time. I strongly recommend that you do not fall into the trap of using scan for condition to get you out of some perceived modeling “jam.” If you *know* what you are doing, you almost never need to use the scan for condition option.

### 8.6.1 Redoing the M/M/1 Model with HOLD/SIGNAL

In this section, we take the very familiar single-server queue and utilize the HOLD/SIGNAL combination to implement it in a more general manner. This modeling will illustrate how the coordination of HOLD and SIGNAL must be carefully constructed. Recall that in a single-server queuing situation, customers arrive and request service from the server. If the server is busy, the customer must wait in the queue; otherwise, the customer enters service. After completing service, the customer then departs.

#### EXAMPLE 8.8 M/M/1 with HOLD/SIGNAL

Assume that we have a single server that performs service according to an exponential distribution with a mean of 0.7 hours. The time between arrival of customers to the server is exponential with mean of 1 hour. If the server is busy, the customer waits in the queue until the server is available. The queue is processed according to an FIFO rule.

*First Solution to Example 8.8:* In this solution, we will not use a RESOURCE module. Instead, we will use a variable, *vServer*, that will represent whether or not the server is busy. When the customer arrives, we will check if the server is busy. If so, the customer will wait until signaled that the server is available.

The pseudo-code for this example is given in Exhibit 8.3. The customer is created and then the time of arrival is recorded in the attribute *myArriveTime*. If the server is busy, *vServer* = 1, then the customer goes into the HOLD queue to wait for an end of service signal. If the server is not busy, the customer makes the server busy and then delays for the service time. After the service is complete, the server is indicated as idle (*vServer* = 0). Then, the queue is checked. If there are any customers waiting, a signal is sent to the HOLD queue to release one customer. Finally, the system time is recorded before the customer departs the system.

Figure 8.41 illustrates the flowchart for the implementation. The model represents the pseudo-code almost verbatim. The new HOLD and SIGNAL modules are shown in Figures 8.42 and 8.43, respectively. Notice that the HOLD module has an option drop down menu that allows the user to select one of the aforementioned hold options. In addition, the *Wait For Value* field indicates the value on which the entities will wait. Here is

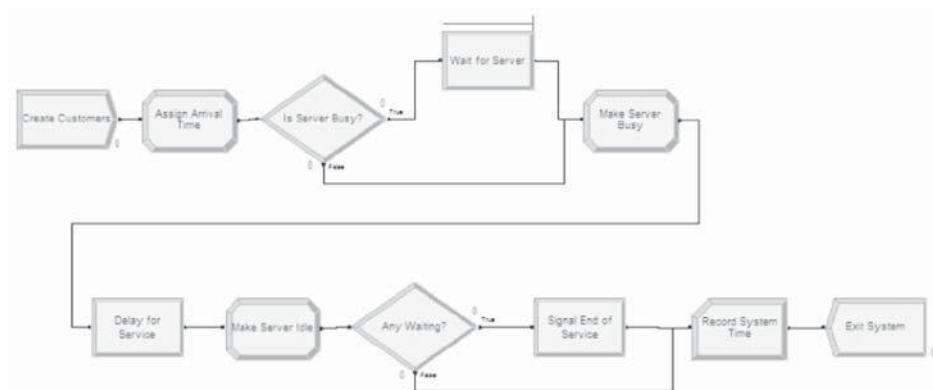


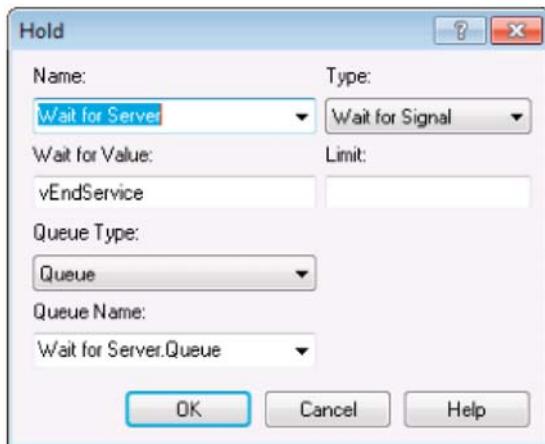
Figure 8.41 First implementation of M/M/1 with HOLD/SIGNAL.

**Exhibit 8.3** Pseudo-Code for M/M/1 with HOLD/SIGNAL

```

CREATE customer every EXPO(1)
ASSIGN myArriveTime = TNOW
DECIDE
    IF vServer == 1 THEN
        HOLD Server Queue
        Wait for End Service Signal
    END HOLD
    END IF
END DECIDE
ASSIGN vServer = 1
DELAY for EXPO(0.7)
ASSIGN vServer = 0
DECIDE
    IF Server Queue is not empty THEN
        SIGNAL End Service
        Signal Limit = 1
    END SIGNAL
    END IF
END DECIDE
RECORD TNOW - myArriveTime
DISPOSE

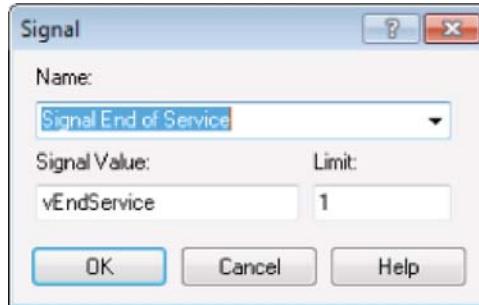
```



**Figure 8.42** HOLD module for first HOLD/SIGNAL M/M/1 example.

it a variable called vEndService. It can be any valid expression that evaluates to an integer. The Limit field (blank in this example) can be used to set to limit how many entities will be released when the signal is received.

Figure 8.43 shows the SIGNAL module. The *Signal Value* field indicates the signal that will be broadcast. The Limit field indicates the number of entities to be released when the signal is completed. In this case, because the server works on one customer at a time, the limit value is set to 1. This causes one customer to be released from the HOLD, whenever the signal is sent. Figure 8.44 presents the result from running the model for 30 replications with warm-up period of 30,000 hours and a run length of 80,000. The results match very closely the theoretical results for this M/M/1 situation.



**Figure 8.43** SIGNAL module for first HOLD/SIGNAL M/M/1 example.

Waiting Time		
	Average	Half Width
Wait for Server.Queue	2.3265	0.03
<b>Other</b>		
Number Waiting		
	Average	Half Width
Wait for Server.Queue	1.6261	0.03
<b>User Specified</b>		
<b>Tally</b>		
Interval		
	Average	Half Width
Record System Time	2.3264	0.02
<b>Time Persistent</b>		
Variable		
	Average	Half Width
vServerBusy	0.6994	0.00

**Figure 8.44** Results for first HOLD/SIGNAL M/M/1 example.

There is one key point to understand this implementation. What would happen if we did not first check if the server was busy before going into the HOLD queue? If we did not do this, then no customers would ever get signaled! This is because, the signal comes at the end of service and no customers would ever reach the SIGNAL. This is a common modeling error when first using the HOLD/SIGNAL combination.

A fundamental aspect of the HOLD/SIGNAL modeling constructs is that entities waiting in the HOLD module are signaled from a SIGNAL module. In the last example, the departing entity signaled the waiting entity to proceed. In the next implementation, we are going to have two types of entities. One entity will represent the customer. The second entity will represent the server. We will use the HOLD/SIGNAL constructs to communicate between the two processes.

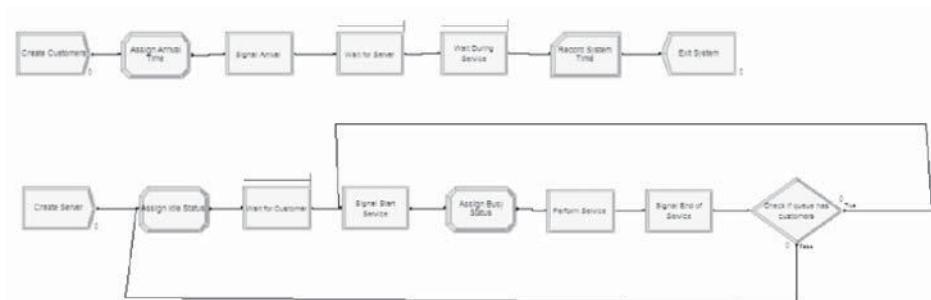
*Second Solution to Example 8.8:* In this implementation, we will have two process flows, one for the customers and one for the server. At first, this will seem very strange. However, this approach will enable a great deal of modeling flexibility. Especially important is the

notion of modeling a resource, such as a server, with its own process. Modeling a resource as an active component of the system (rather than just passively being called) provides great flexibility. Flexibility that will serve you well when approaching some complicated modeling situations. The key to using two process flows for this modeling is to communicate between the two processes via the use of signals.

Figure 8.45 presents the model for this second implementation. Note the two process flows in the figure. The flows follow very closely the pseudo-code provided in Exhibit 8.4. The file *MM1ModelViaWaitSignal.doe* contains the details.

In Exhibit 8.4, the first CREATE module creates customers according to the exponential time between arrivals and then the time of arrival is noted in order to later record the system time. Next, the customer signals that there has been an arrival. Think of it as the customer ringing the customer service bell at the counter. The customer then goes into the HOLD to await for the beginning of service. If the server was busy when the signal went off, the customer just continues to wait. If the server was idle (waiting for a customer to arrive), then the server moves forward in their process to signal the customer to start service. When the customer gets the start service signal, the customer moves immediately into another HOLD queue to wait while the server is performing the service. When the end of service signal occurs, the customer departs after recording the system time.

The second CREATE module, creates a single entity that represents the server. Essentially, the server will cycle through being idle and busy. After being created, the server sets its status to idle (0) and then goes into a HOLD queue to await for the first customer's arrival. When the customer arrives, the customer sends a signal. The server can then proceed with service. First, the server signals the customer that service is beginning. This allows the customer to proceed to the hold queue for service. Then, the server delays for the service time. After the service time is complete, the server signals the customer that the end of service has occurred. This allows the customer to depart the system. After signaling the end of service, the server checks if there are any waiting customers; if there are waiting customers, the entity representing the server is sent to Label B to signal a start of service. If there are no customers waiting, then the entity representing the server is sent to Label A to wait in a hold queue for the next customer to arrive.



**Figure 8.45** Second implementation of M/M/1 with HOLD/SIGNAL.

---

**Exhibit 8.4** Solution 2 for M/M/1 with HOLD/SIGNAL

---

```

CREATE customer every EXPO(1)
ASSIGN myArriveTime = TNOW
SIGNAL Arrival
    Signal Value = vArrivalSignal
END SIGNAL
HOLD Server Queue
    Wait for Begin Service Signal
END HOLD
HOLD In Service Queue
    Wait for End Service Signal
END HOLD
RECORD TNOW - myArriveTime
DISPOSE

CREATE 1 server at time 0.0
Label: A
ASSIGN vServerStatus = 0
HOLD Wait for Customer
    Wait for Value = vArrivalSignal
END HOLD
Label: B
SIGNAL Start Service
    Signal Value = vStartService
    Signal Limit = 1
END SIGNAL
ASSIGN vServerStatus = 1
DELAY for EXPO(0.7)
SIGNAL End Service
    Signal Value = vEndService
    Signal Limit = 1
END SIGNAL
DECIDE
    IF Server Queue is not empty THEN
        Send to Label B:
    ELSE
        Send to Label A:
    END IF
END DECIDE

```

---

This signaling back and forth must seem overly complicated. However, this example illustrates how two entities can communicate with signals. In addition, the notion of modeling a server as a process should open up some exciting possibilities. The modeling of a server as a process permits very complicated resource allocation rules to be embedded in a model.

The last two sections examined small queuing systems, involving one or two queues. In order to model more complex queuing systems, we must be able to easily have a large number of queues, link the queues together, and route entities between the queues. This brings up the very important topic of networks of queues, which is examined in the next section.

## 8.7 NETWORKS OF QUEUING STATIONS

A network of queues can be thought of as a set of stations, where each station represents a service facility. In the simple case, each service facility might have a single-queue feeding into a resource. In the most general case, customers may arrive to any station within the network from outside the system. Customers then proceed from station to station to receive service. After receiving all of their service requirements, the customers depart the system.

In general, the customers do not take the same path through the network and for a given customer type, the path may be deterministic or stochastic in nature. For example, consider the case of a job shop manufacturing setting. Each product that is manufactured may require a different set of manufacturing processes, which are supplied by particular stations (e.g., drilling and milling). As another example, consider a telecommunications network where packets are sent from some origin to some destination through a series of routers. In each of these cases, understanding how many resources to supply so that the customer can efficiently traverse the network at some minimum cost is important. As such, queuing networks are an important component of the efficient design of manufacturing, transportation, distribution, telecommunication, computer, and service systems. Figure 8.46 illustrates the concept of a network of queues for producing a vacuum cleaner.

The analytical treatment of the theory associated with networks of queues has been widely examined and remains an active area for theoretical research. It is beyond the scope of this text to discuss the enormous literature on queuing networks. The interested reader is referred to the following texts as a starting point, Gross and Harris [1998], Kelly [1979], Buzacott and Shanthikumar [1993], or Bolch et al. [2006].

A number of examples that can be considered queuing networks (e.g., the Tie-Dye T-Shirts and the LOTR's Ring Making examples) have already been examined. The purpose of this section is to introduce some of the constructs within Arena<sup>TM</sup> that facilitate the simulation of queuing networks. Since a queuing network involves the movement of entities between stations, the use of the STATION, ROUTE, and SEQUENCE modules from the Advanced Transfer template panel will be emphasized.

### EXAMPLE 8.9 Testing and Repair Shop

Consider a test and repair shop for computer parts (e.g., circuit boards and hard drives). The system consists of an initial diagnostic station through which all newly arriving

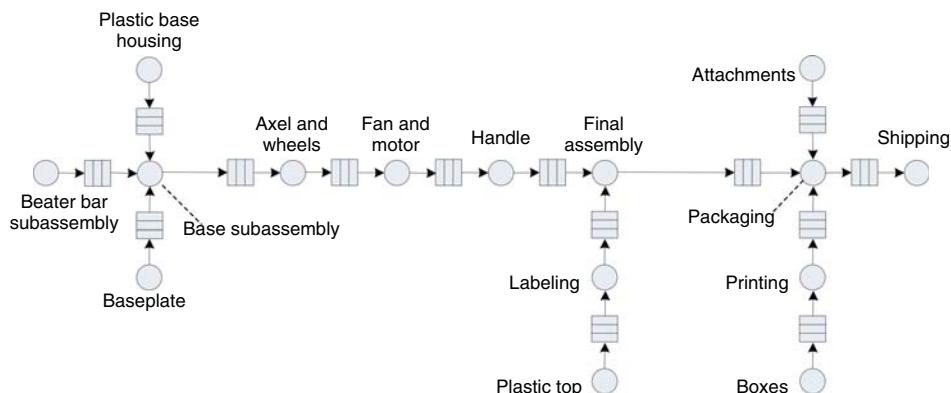


Figure 8.46 A network of queues.

parts must be processed. Currently, newly arriving parts arrive according to a Poisson arrival process with a mean rate of 3 per hour. The diagnostic station consists of two diagnostic machines that are fed the arriving parts from a single queue. Data indicates that the diagnostic time is quite variable and follows an exponential distribution with a mean of 30 minutes. Based on the results of the diagnostics, a testing plan is formulated for the parts. There are currently three testing stations, 1, 2, and 3, which consist of one machine each. The testing plan consists of an ordered sequence of testing stations that must be visited by the part prior to proceeding to a repair station. Because the diagnosis often involves similar problems, there are common sequences that occur for the parts. The company collected extensive data on the visit sequences for the parts and found that the following sequences constituted the vast majority of test plans for the parts.

Test Plan	Percentage of Parts (%)	Sequence
1	25	2,3,2,1
2	12.5	3,1
3	37.5	1,3,1
4	25	2,3

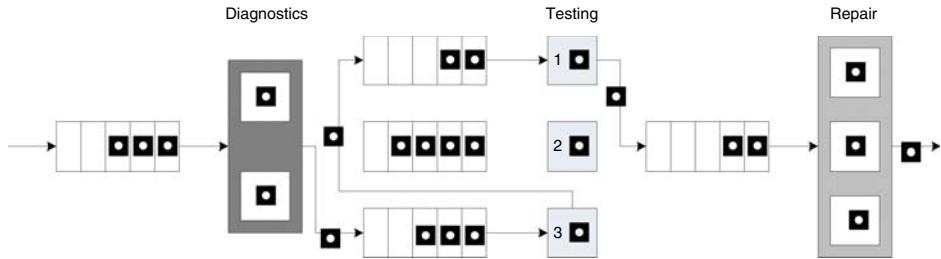
For example, 25% of the newly arriving parts follow test plan 1, which consists of visiting test stations 2, 3, 2, and 1 prior to proceeding to the repair station.

The testing of the parts at each station takes time that may depend upon the sequence that the part follows. That is, while parts that follow both test plans 1 and 3 visit test station 1, data shows that the time spent processing at the station is not necessarily the same. Data on the testing times indicate that the distribution is well modeled with a log-normal distribution with mean,  $\mu$ , and standard deviation,  $\sigma$ , in minutes. The following table presents the mean and standard deviation for each of the testing time distributions by each station in the test plan.

Test Plan	Testing Time Parameters	Repair Time Parameters
1	(20,4.1), (12,4.2), (18,4.3), (16,4.0)	(30,60,80)
2	(12,4), (15,4)	(45,55,70)
3	(18,4.2), (14,4.4), (12,4.3)	(30,40,60)
4	(24,4), (30,4)	(35,65,75)

For example, the first pair of parameters, (20, 4.1), for test plan 1 indicates that the testing time at test station 2 has a lognormal distribution with mean,  $\mu = 20$ , and standard deviation,  $\sigma = 4.1$  minutes.

The repair station has three workers who attempt to complete the repairs based on the tests. The repair time also depends on the test plan that the part has been following. Data indicates that the repair time can be characterized by a triangular distribution with the minimum, mode, and maximum as specified in the previous table. After the repairs, the parts leave the system. When the parts move between stations assume that there is always a worker available and that the transfer time takes between 2 and 4 minutes uniformly distributed. Figure 8.47 illustrates the arrangement of the stations and the flow of the parts following Plan 2 in the test and repair shop.



**Figure 8.47** Test and repair shop.

The company is considering accepting a new contract that will increase the overall arrival rate of jobs to the system by 10%. They are interested in understanding where the potential bottlenecks are in the system and in developing alternatives to mitigate those bottlenecks so that they can still handle the contract. The new contract stipulates that 80% of the time, the testing and repairs should be completed within 480 minutes. The company runs two shifts each day for each 5-day work week. Any jobs not completed at the end of the second shift are carried over to first shift of the next working day. Assume that the contract is going to last for 1 year (52 weeks). Build a simulation model that can assist the company in assessing the risks associated with the new contract.

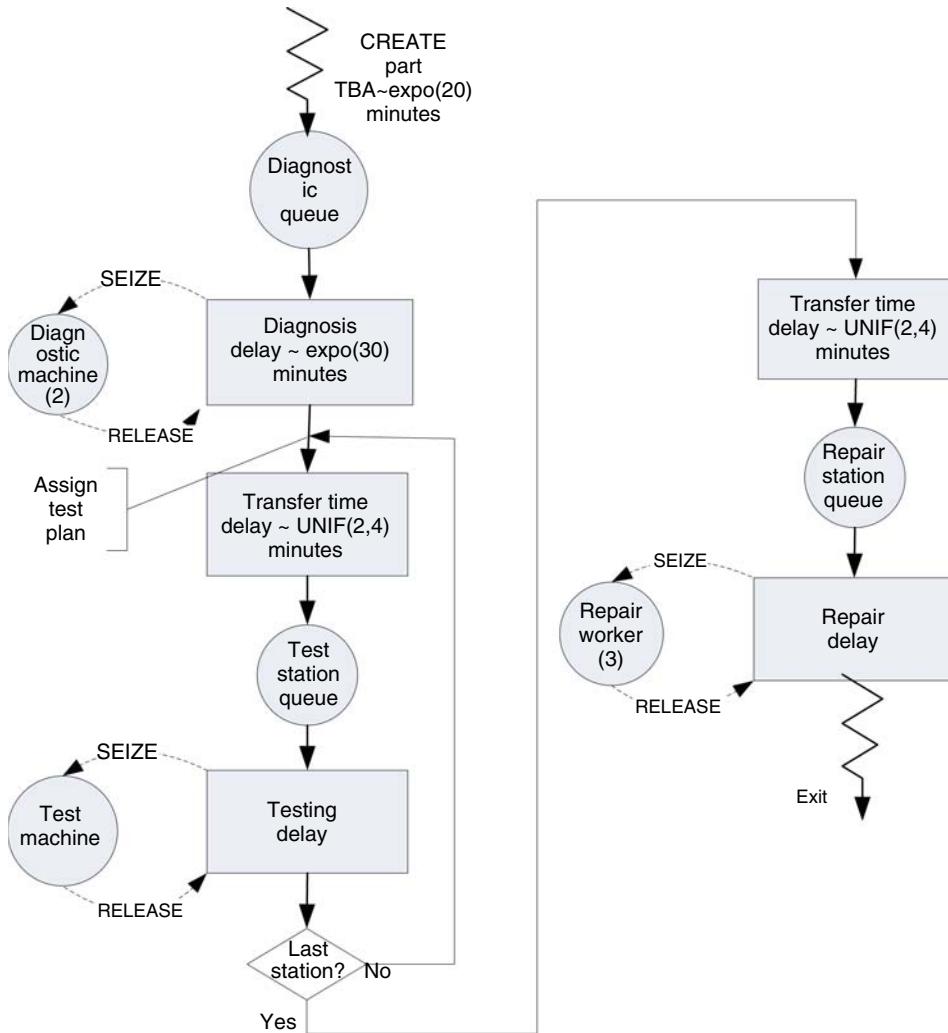
*Solution to Example 8.9:* Before implementing the model in Arena™, you should prepare by conceptualizing the process flow. Figure 8.48 illustrates the activity diagram for the test and repair system. Parts are created and flow first to the diagnostic station where they seize a diagnostic machine while the diagnostic activity occurs. Then, the test plan is assigned. The flow for the visitation of the parts to the test station is shown with a loop back to the transfer time between the stations. It should be clear that the activity diagram is representing any of the three test stations. After the final test station in the test plan has been visited, the part goes to the repair station, where one of three repair workers is seized for the repair activity. After the repair activity, the part leaves the system.

Exhibit 8.5 presents the pseudo-code for the test and repair model. This is a straightforward representation of the flow presented in the activity diagram. From the activity diagram and the pseudo-code, it should be clear that with the current Arena™ modeling constructs, you should be able to model this situation. In order to model the situation, you need some way to represent where the part is currently located in the system (e.g., the current station). Second, you need some way to indicate where the part should go to next. And finally, you need some way to model the transfer and the time of the transfer of the part between stations. This type of modeling is very common. Because of this, there are special modules that are specifically designed to handle situations like this.

The pseudo-code introduces some new keywords: SEQUENCE, STATION, and ROUTE. The next section introduces these very useful constructs.

### 8.7.1 STATION, ROUTE, and SEQUENCE Modules

To model the Test and Repair Shop, three new modules and the concept of transferring entities need to be covered. Entities typically represent the things that are flowing through the system. In previously examined models, the *direct connect* method of transferring entities between modules within the model has been used. With the direct connect method,



**Figure 8.48** Activity diagram for test and repair system.

two modules are directly connected via a connection line within the model window and the entity flow along the connection line between the modules. The modeling constructs within this section allow entities to move between modules without a connection line.

*Entity transfer* refers to the various ways by which entities can move between modules. The STATION and ROUTE modules facilitate the transfer of entities between stations with a transfer delay. The SEQUENCE module defines a list of stations to be visited.

**STATION** The STATION module represents a named location to which entities can be transferred. Initially, one can think of stations as the label part of the Go To Label construct found in standard programming languages; however, stations are more powerful than a simple label. Stations can be placed in sets and held within sequences. Stations are also necessary for mapping model logic transfers to a physical (spatial) representation of the world.

**Exhibit 8.5** Pseudo-Code for Test and Repair System

---

```

CREATE part
PROCESS Diagnostics
    SEIZE 1 diagnostic machine
    DELAY for diagnostic time
    RELEASE diagnostic machine
END PROCESS
ASSIGN
    determine test plan type
    determine SEQUENCE for test plan type
END ASSIGN
ROUTE for transfer time by SEQUENCE to STATION Test

STATION Test
PROCESS Testing
    SEIZE appropriate test machine
    DELAY for testing time
    RELEASE test machine
END PROCESS
DECIDE
    IF not at last station THEN
        ROUTE for transfer time by SEQUENCE to STATION Test
    ELSE
        ROUTE for transfer time by SEQUENCE to STATION Repair
    END IF
END DECIDE

STATION Repair
PROCESS Repair Process
    SEIZE repair worker from repair worker set
    DELAY for repair time
    RELEASE repair worker
END PROCESS
RECORD statistics
DISPOSE

```

---

**ROUTE** The ROUTE module causes an entering entity to be transferred to a station with a possible time delay that accompanies the transfer. The entity leaves the route module and after the specified time delay reappears in the model at the specified station.

**SEQUENCE** The SEQUENCE module is a data module that defines an ordered list of stations. The list represents the natural order in which the stations will be visited, if transferred using the ROUTE module with the *By Sequence* option. In addition to defining a list of stations, a SEQUENCE permits the listing of a set of assignments (e.g., ASSIGN modules) to be executed when the transfer occurs.

Entities have a number of special-purpose attributes that are useful when modeling with the SEQUENCE, STATION, and ROUTE modules. The *Entity.Station* and *Entity.CurrentStation* keep track of the location of the entity within the model. The attribute *Entity.CurrentStation* is updated to the current station whenever the entity passes through the STATION module. This attribute is not user assignable but can be used (read) in the model. It will return the station number for the current station of the entity or 0 if the entity is not currently at a station. In addition, every entity has an *Entity.Station*

attribute, which returns the entity's station or destination. The *Entity.Station* attribute is user assignable. It is (automatically) set to the intended destination station when an entity is transferred (e.g., via a ROUTE module). It will remain equal to the current station after the transfer or until either changed by the user or affected by another transfer type module (e.g., ROUTE module). Thus, the modules attached to a STATION module are conceptually at the station location.

In many modeling contexts, entities will follow a specific path through the system. In a manufacturing job shop, this is often called the process plan. In a bus system, this is called a bus route. In the test and repair system, this is referred to as the test plan. To model a specify path through the system, the SEQUENCE module can be used. A sequence consists of an ordered list of *job steps*. Each job step must indicate the STATION associated with the step and may indicate a series of assignments that must take place when the entity reaches the station associated with the job step. Each job step can have an optional name and can give the name of the next step in the sequence. Thus, a sequence is built by simply providing the list of stations that must be visited.

Each entity has a number of special-purpose attributes that facilitate the use of sequences. The *Entity.Sequence* attribute holds the sequence that the entity is currently following or 0 if no sequence has been assigned. The ASSIGN module can be used to assign a specific sequence to an entity. In addition, the entity has the attribute *Entity.JobStep*, which indicates the current step within the sequence that the entity is executing. *Entity.JobStep* is user assignable and is automatically incremented when a transfer type module (e.g., ROUTE) is used with the *By Sequence* option. Finally, the attribute *Entity.PlannedStation* is available and represents the number of the station associated with the next job step in the sequence. *Entity.PlannedStation* is not user assignable. It is automatically updated whenever *Entity.Sequence* or *Entity.JobStep* changes, or whenever the entity enters a station.

In the test and repair example, STATION modules will be associated with the diagnostic station, each of three test stations, and with the repair station. The work performed at each station will be modeled with a PROCESS module using the (SEIZE, DELAY, and RELEASE) option. The four different test plans will be modeled with four sequences. Finally, ROUTE modules using the *By Sequence* transfer option will be used to transfer the entities between the stations with a UNIF(2,4) transfer delay time distribution in minutes. Other than the new modules for entity transfer, this model is similar to previous models. This model can be built by following these steps:

1. Define five different resources (*DiagnosticMachine*, *TestMachine1*, *TestMachine2*, *TestMachine3*, *RepairWorkers*) with capacity (2, 1, 1, 1, 3), respectively.
2. Define two variables (*vContractLimit*, *vMTBA*) with initial values (480 and 20), respectively. These variables represent the length of the contract and the mean time between arrivals, respectively.
3. Define three expressions (*eDiagnosticTime*, *eTestPlanCDF*, and *eRepairTimeCDFs*). *eDiagnosticTime* should be *expo(30)*, *eTestPlanCDF* should be *DISC(0.25, 1, 0.375, 2, 0.75, 3, 1.0, 4)*. In this distribution, 1, 2, 3, 4 represents the four test plans. Finally, *eRepairTimeCDFs* should be an arrayed expression with four rows. Each row should specify a triangular distribution according to the information given concerning the repair time distributions (e.g., *TRIA(30,60,80)*).
4. Use the SEQUENCE module on the Advanced Transfer panel to define four different sequences. This is illustrated in Figure 8.49. First, double click on the SEQUENCE module to add a new sequence (row). The first sequence is called *TestPlan1Seq*. Then,



Figure 8.49 Defining sequences, job steps, and assignments.

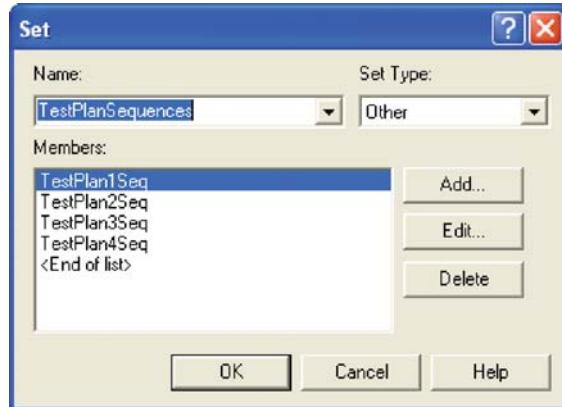
add job steps to the sequence by clicking on the Steps button. Figure 8.49 shows the five-job steps for test plan 1 (*TestStation2*, *TestStation3*, *TestStation2*, *TestStation1*, and *RepairStation*). Typing in a station name defines a station for use in the model window. Since every part must visit the repair station before exiting the system, *RepairStation* has been listed last in all the sequences.

For each job step, define an assignment that will happen when the entity is transferred to the step. In the test and repair system, the testing time depends upon the job step. Thus, an attribute, *myTestingTime*, can be defined so that the value from the pertinent lognormally distributed test time distribution can be assigned. In the case illustrated, *myTestingTime* will be set equal to a random number from a LOGN(20, 4.1) distribution, which represents the distribution for test station 2 on the first test plan. The *myTestingTime* attribute is used to specify the delay time for each of the PROCESS modules that represent the testing processes.

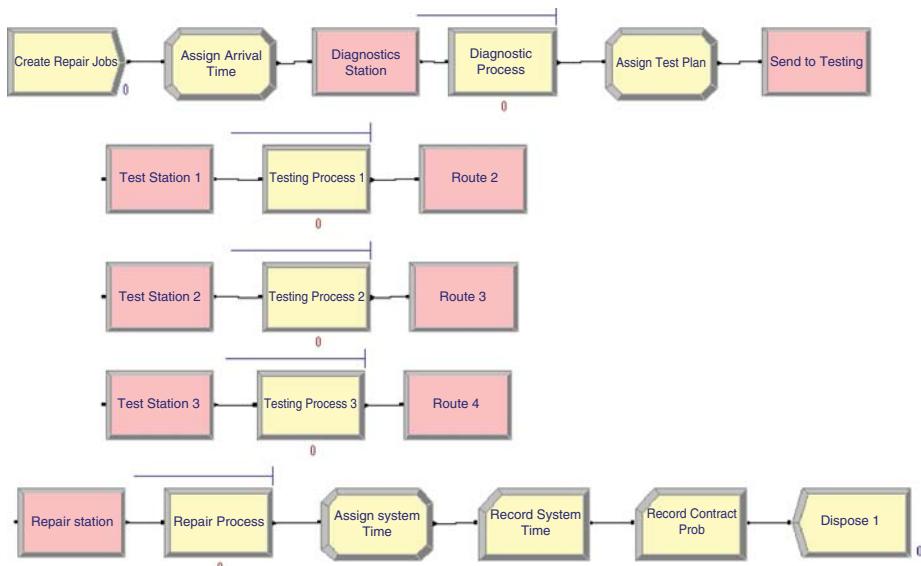
- Finally, define a set to hold the sequences so that they can be randomly assigned to the parts after they visit the diagnostic machine. To define a Sequence set, you must use the Advance Set module on the Advance Process panel. Each of the sequences should be listed in the set as shown in Figure 8.50. The set type should be specified as *Other*. Unfortunately, the build expression option is not available here and each sequence name must be carefully typed into the dialog box. The order of the sequences is important.

Now that all the data modules have been specified, you can easily build the model using the flowchart modules. Figure 8.51 presents an overview of the model. The pink-colored modules in the figure are the STATION and ROUTE modules. A CREATE module is used to generate the parts needing testing and repair with a mean time between arrivals of 20 minutes exponentially distributed. Then, the entity proceeds through an ASSIGN module where the attribute, *myArriveTime*, is set to TNOW. This will be used to record the job's system time. The next module is a STATION module that represents the diagnostic station, see Figure 8.52.

After passing through the STATION module, the entity goes through the diagnostic process. Following the diagnostic process, the part is assigned a test plan. Figure 8.53 shows that the test plan expression holding the DISC distribution across the test plans is used to assign a number 1, 2, 3, or 4 to the attribute *myTestPlan*. Then, the attribute is used to



**Figure 8.50** Advanced set to hold sequences.



**Figure 8.51** Overview of test and repair model.

select the appropriate sequence from our previously defined set of test plan sequences. The sequence returned from the set is assigned to the special-purpose attribute, *Entity.Sequence*, so that the entity can now follow this sequence.

The part then enters the ROUTE module for sending the parts to the testing stations. Figure 8.54 illustrates the ROUTE module. This module allows a time delay in the route time field and allows the user to select the method by which the entity will be transferred. Choosing the *By Sequence* option indicates that the entity should use its assigned sequence (via the *Entity.Sequence* attribute) to determine the destination station for the route. The entity's sequence and its current job step are used. When the entity goes into the ROUTE module, *Entity.JobStep* is incremented to the next step. The entity's *Entity.Station* attribute is set equal to the station associated with the step and all attributes associated with the



Figure 8.52 Diagnostics station STATION module

Assignments			
	Type	Attribute Name	New Value
1	Attribute	myTestPlan	eTestPlanCDF
2	Attribute	Entity.Sequence	TestPlanSequences(myTestPlan)

Double-click here to add a new row.

Figure 8.53 Test plan assignments.

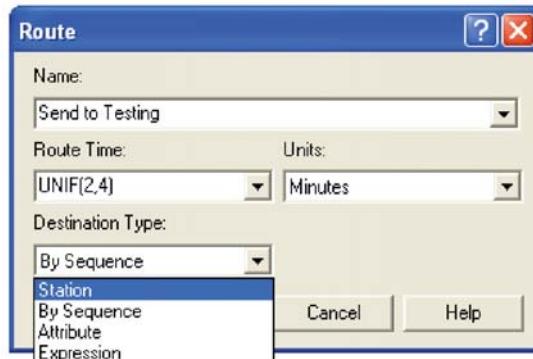


Figure 8.54 ROUTE module.

step are executed. Then, the entity is transferred (starts the delay associated with the transfer). After the entity completes the transfer and enters the destination station, the entity's *Entity.CurrentStation* attribute is updated.

In the example, the part is sent to the appropriate station on its sequence. Each of the stations used to represent the testing stations follow the same pattern (STATION, PROCESS

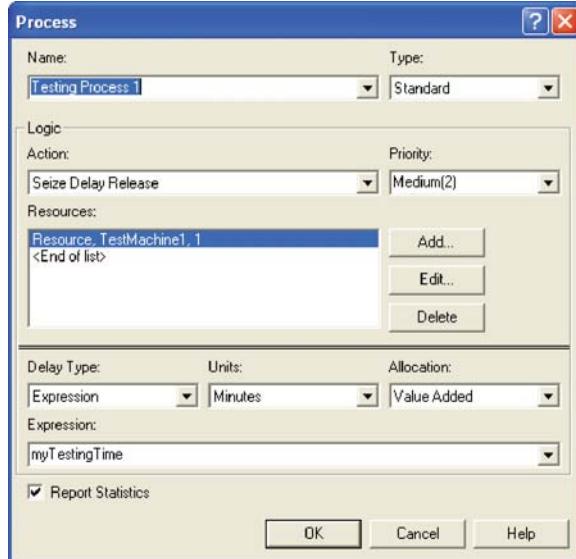


Figure 8.55 PROCESS module for testing process.

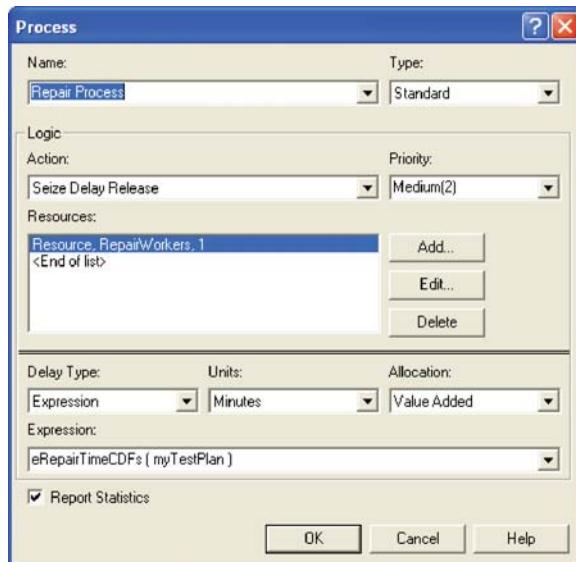


Figure 8.56 Repair process module.

(seize, delay, release), and ROUTE). The PROCESS module uses the attribute *myTestingTime* to determine the delay time for the testing process. This attribute was set when the entity's job step attributes were executed (Figure 8.55).

After proceeding through its testing plan, the part is finally routed to the *RepairStation*, since it was the station associated with the last job step. At the repair station, the part goes through its repair process by using the expression *eRepairTimeCDFs* and its attribute, *myTestPlan*, as shown in Figure 8.56.

The ASSIGN module after the repair process module simply computes the entity's total system time in the attribute *mySysTime*, so that the following two RECORD modules can compute the appropriate statistics as indicated in Figure 8.57.

Now the model is ready to set up and run. According to the problem, the purpose of the simulation is to evaluate the risk associated with the contract. The life of the contract is specified as 1 year ( $52 \text{ weeks} \times 5 \text{ days/week} \times 2 \text{ shifts/day} \times 480 \text{ minutes/shift} = 249,600 \text{ minutes}$ ). Since the problem states that any jobs not completed at the end of a shift are carried over to the next working day, it is as if there are 249,600 minutes or 4160 hours of continuous operation available for the contract. This is also a terminating simulation since performance during the life of the contract is the primary concern. The only other issue to address is how to initialize the system. The analysis of the two situations (current contract vs contract with 10% more jobs) can be handled via a relative comparison. Thus, to perform a *relative* comparison, you need to ensure that both alternatives start under the same initial conditions. For simplicity, assume that the test and repair shop starts each contract alternative under empty and idle conditions. Let us assume that 10 replications of 4160 hours will be sufficient as illustrated in Figure 8.58.

Record - Basic Process						
	Name	Type	Value	Record into Set	Tally Name	
1	Record System Time	Expression	mySysTime	<input type="checkbox"/>	SystemTimeStat	
2	Record Contract Prob	Expression	mySysTime < vContractLimit	<input type="checkbox"/>	ProbLTContractLimit	

Figure 8.57 RECORD modules.

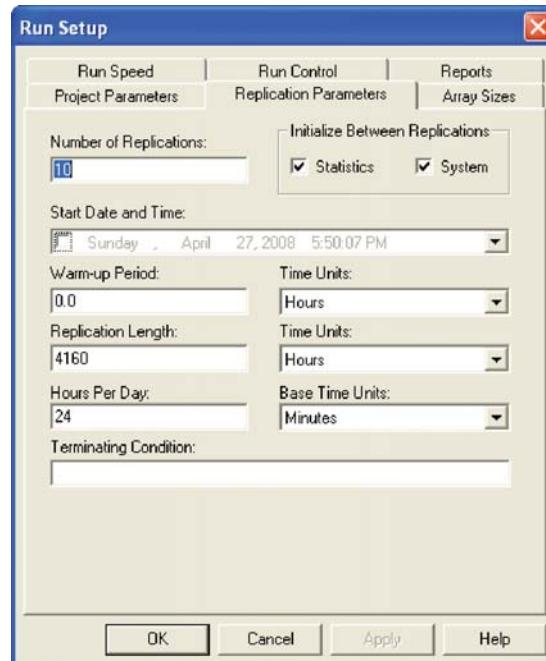


Figure 8.58 Test and repair shop Run Setup specification.

User Specified		
Tally		
Expression	Average	Half Width
ProbLTContractLimit	0.8251	0.02
SystemTimeStat	350.93	11.00

**Figure 8.59** User-defined statistics for current contract.

As shown in Figure 8.59, for the current situation, the probability that a job completes its testing and repair within 480 minutes is about 82%. The addition of more jobs should increase the risk of not meeting the contract specification. You are asked to analyze the new contract's risks and make a recommendation to the company on how to proceed within the exercises.

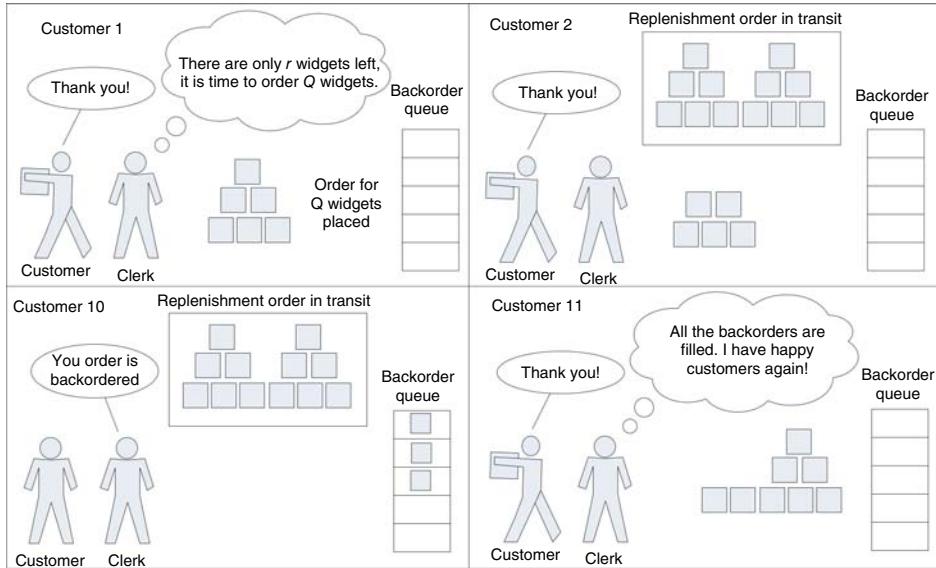
In the test and repair example, the time that it took to transfer the parts between the stations was a simple stochastic delay (e.g., UNIF (2,4) minutes). The STATION, ROUTE, and SEQUENCE modules make the modeling of entity movement between stations in this case very straightforward. In some systems, the modeling of the movement is much more important because material handling devices (e.g., people to carry the parts, fork lifts, and conveyors) may be required during the transfer. These will require an investigation of the modules within the Advanced Transfer panel. This topic will be taken up in Chapter 9.

To finish up this chapter, another classic situation in which customers may wait to have their requests fulfilled will be examined: inventory systems.

## 8.8 INVENTORY SYSTEMS

In an inventory system, there are units of an item (e.g., computer printers) for which customers make demands. If the item is available (in stock), then the customer can receive the item and depart. If the item is not on hand when a demand from a customer arrives, then the customer may depart without the item (i.e., lost sales) or the customer may be placed on a list for when the item becomes available (i.e., backordered). In a sense, the item is like a resource that is consumed by the customer. Unlike the previous notions of a resource, inventory can be replenished. The proper control of the replenishment process is the key to providing adequate customer service. There are two basic questions that must be addressed when controlling the inventory replenishment process: (i) When to order? and (ii) How much to order?. If the system does not order enough or does not order at the right time, the system may not be able to fill customer demand in a timely manner. Figure 8.60 illustrates a simple inventory system.

There are a number of different ways to manage the replenishment process for an inventory item. These different methods are called *inventory policies*. An inventory control policy must determine (at the very least) when to place a replenishment order and how much to order. This section examines the use of a reorder point ( $r$ ), reorder quantity ( $Q$ ) inventory policy. This is often denoted as an  $(r, Q)$  inventory policy. The modeling of a number of other inventory control policies will be explored as exercises. After developing a basic understanding of how to model an  $(r, Q)$  inventory system, the modeling can be expanded to study *supply chains*. A supply chain can be thought of as a network of locations that hold inventory in order to satisfy end customer demand.



**Figure 8.60** A simple  $(r, Q)$  inventory system where  $r = 6$  and  $Q = 12$ .

Just like the topic of queuing systems, the topic of inventory systems has been well studied. In what follows, the analytical treatment of the  $(r, Q)$  inventory policy will be discussed. A full exposition of the topic of inventory systems is beyond the scope of this text, but the reader can refer to a number of texts within the area, such as Hadley and Whitin [1963], Axsäter [2006], Silver et al. [1998], or Zipkin [2000] for more details. The reader interested in supply chain modeling might refer to Nahmias [2001], Askin and Goldberg [2002], Chopra and Meindl [2007], or Ballou [2004].

### 8.8.1 Modeling an $(r, Q)$ Inventory Control Policy

This section develops a model of a continuous review  $(r, Q)$  inventory system with backordering. In a continuous review  $(r, Q)$  inventory control system, demand arrives according to some stochastic process. When a demand (customer order) occurs, the amount of the demand is determined, and then the system checks for the availability of stock. If the stock on hand is enough for the order, the demand is filled and the quantity on hand is decreased. On the other hand, if the stock on hand is not enough to fill the order, the entire order is backordered. The backorders are accumulated in a queue and they will be filled after the arrival of a replenishment order. Assume for simplicity that the backorders are filled on a FCFS basis. The inventory position (inventory on hand plus on-order minus backorders) is checked each time after a regular customer demand and the occurrence of a backorder. If the inventory position reaches or falls under the reorder point, a replenishment order is placed. The replenishment order will take a possibly random amount of time to arrive and fill any backorders and increase the on-hand inventory. The time from when a replenishment order is placed until the time that it arrives to fill any backorders is often called the lead-time for the item.

There are three key state variables that are required to model this situation. Let  $I(t)$ ,  $IO(t)$ , and  $BO(t)$  be the amount of inventory on hand, on order, and backordered, respectively, at time  $t$ . The net inventory,  $IN(t) = I(t) - BO(t)$ , represents the amount of inventory (positive or negative). Notice that if  $I(t) > 0$ , then  $BO(t) = 0$  and that if  $BO(t) > 0$ , then  $I(t) = 0$ . These variables compose the inventory position, which is defined as

$$IP(t) = I(t) + IO(t) - BO(t)$$

The inventory position represents the current amount on hand,  $I(t)$ ; the amounted back-ordered,  $BO(t)$ ; and the amount previously ordered,  $IO(t)$ . Thus, when placing an order, the inventory position can be used to determine whether or not a replenishment order needs to be placed. Since  $IO(t)$  is included in  $IP(t)$ , the system will only order when outstanding orders are not enough to get  $IP(t)$  above the reorder point.

In the continuous review  $(r, Q)$  policy, the inventory position must be checked against the reorder point as demands arrive to be filled. After filling (or backordering) a demand, either  $I(t)$  or  $BO(t)$  will have changed (and thus  $IP(t)$  will change). If  $IP(t)$  changes, it must be checked against the reorder point. If  $IP(t) \leq r$ , then an order for the amount  $Q$  is placed.

For simplicity, assume that each demand that arrives is for 1 unit. This simplifies how the order is processed and the processing of any backorders. The pseudo-code for this situation is given in Exhibit 8.6.

Referring to the exhibit, the entity being created is a customer demand. The amount of the demand should be an attribute *myAmtDemanded* of the entity having value 1. After the customer arrives, the customer is sent to order fulfillment. Notice the use of a label to indicate the order-filling logic.

At order fulfillment, we need to check if there is inventory available to satisfy the demand. If the amount required can be satisfied,  $I(t) \geq \text{myAmtDemanded}$ . Then, the on-hand inventory is decremented by the amount of the demand. If the demand cannot be satisfied,  $I(t) < \text{myAmtDemanded}$ . In this case, the amount waiting in the back order queue,  $BO(t)$ , is incremented by the amount demanded. Then, a duplicate is made so that the duplicate can be sent to the backorder queue.

In either case of filling, the demand or not filling the demand, the inventory position must be updated. This is because either  $I(t)$  or  $BO(t)$  changed. Since the inventory position changed, we must check if a replenishment order is necessary. The entity is sent to the replenishment logic.

At the replenishment logic, the inventory position is checked against the reorder point. If the inventory position is less than or equal to the reorder point, an order is placed. If no order is placed, the entity skips over the order placement logic and is disposed. If an order is placed, the on-order variable is incremented by the reorder quantity and the delay for the lead-time started. Once the lead-time activity is completed, the on-order and on-hand variables are updated and a signal is sent to the backorder queue. A signal is sent to the back-order queue so that if there are any demands waiting in the backorder queue, the demands can try to be filled.

The key performance measures for this type of system are the average amount of inventory on hand, the average amount of inventory backordered, the percentage of time that the system is out of stock, and the average number of orders made per unit time.

**Exhibit 8.6** Pseudo-Code for  $(r, Q)$  Inventory Model

---

```

CREATE demand
ASSIGN myAmtDemand = 1
Send to Label: Order Fulfillment

Label: Order Fulfillment
DECIDE
    IF  $I(t) \geq$  myAmtDemand THEN // fill the order
        ASSIGN  $I(t) = I(t) -$  myAmtDemand
    ELSE// handle back order
        ASSIGN  $BO(t) = BO(t) +$  myAmtDemand
        SEPARATE Duplicate 1 entity
            Send duplicate to Label: Handle Back Order
        END SEPARATE
    END IF
     $IP(t) = I(t) + IO(t) - BO(t)$ 
    Send to Label: Replenishment Ordering
END DECIDE

Label: Handle Back Order
HOLD Back order queue: wait for signal
    On Signal: release all from queue
END HOLD
ASSIGN
     $BO(t) = BO(t) -$  myAmtDemand
END ASSIGN
Send to Label: Order Fulfillment

Label: Replenishment Ordering
IF  $IP(t) \leq r$  THEN
    ASSIGN  $IO(t) = IO(t) + Q$  // Place the order
    DELAY for lead time
    ASSIGN
         $IO(t) = IO(t) - Q$  // Receive the order
         $I(t) = I(t) + Q$ 
    END ASSIGN
    SIGNAL Hold back order queue
END IF
DISPOSE

```

---

Let us discuss these performance measures before indicating how to collect them within a simulation. The average inventory on hand and the average amount of inventory backordered can be defined as follows:

$$\bar{I} = \frac{1}{T} \int_0^T I(t) dt$$

$$\bar{BO} = \frac{1}{T} \int_0^T BO(t) dt$$

As can be seen from the definitions, both  $I(t)$  and  $BO(t)$  are time-based variables and their averages are time averages. Under certain conditions as  $T$  goes to infinity, these time averages will converge to the steady-state performance for the  $(r, Q)$  inventory model. The

percentage of time that the system is out of stock can be defined based on  $I(t)$  as follows:

$$SO(t) = \begin{cases} 1 & I(t) = 0 \\ 0 & I(t) > 0 \end{cases}$$

$$\overline{SO} = \frac{1}{T} \int_0^T SO(t) dt$$

Thus, the variable  $SO(t)$  indicates whether or not there is no stock on hand at any time. A time-average value for this variable can also be defined and interpreted as the percentage of time that the system is out of stock. One minus  $\overline{SO}$  can be interpreted as the percentage of time that the system has stock on hand. Under certain conditions (but not always), this can also be interpreted as the fill rate of the system (i.e., the fraction of demands that can be filled immediately). Let  $Y_i$  be an indicator variable that indicates 1 if the  $i$ th demand is immediately filled (without back ordering) and 0 if the  $i$ th demand is backordered upon arrival. Then, the fill rate is defined as follows.

$$\overline{FR} = \frac{1}{n} \sum_{i=1}^n Y_i$$

Thus, the fill rate is just the average number of demands that are directly satisfied from on-hand inventory. The variables  $\overline{SO}$  and  $\overline{FR}$  are measures of customer service.

To understand the cost of operating the inventory policy, the average number of replenishment orders made per unit time or the *average order frequency* needs to be measured. Let  $N(t)$  be the number of replenishment orders placed in  $(0, t]$ , then the average order frequency over the period  $(0, t]$  can be defined as

$$\overline{OF} = \frac{N(T)}{T}$$

Notice that the average order frequency is a rate (units/time).

In order to determine the best settings of the reorder point and reorder quantity, we need an objective function that trades off the key performance measures within the system. This can be achieved by developing a total cost equation on a per time period basis. Let  $h$  be the holding cost for the item in terms of \$/unit/time. That is, for every unit of inventory held, we accrue  $h$  dollars per time period. Let  $b$  be the backorder cost for the item in terms of \$/unit/time. That is, for every unit of inventory backordered, we accrue  $b$  dollars per time period. Finally, let  $k$  represent the cost in dollars per order whenever an order is placed. The settings of the reorder point and reorder quantity depend on these cost factors. For example, if the cost per order is very high, then we should not want to order very often. However, this means that we would need to carry a lot of inventory to prevent a high chance of stocking out. If we carry more inventory, then the inventory holding cost is high.

The average total cost per time period can be formulated as follows:

$$\overline{TC} = k\overline{OF} + h\overline{I} + b\overline{BO}$$

A discussion of the technical issues and analytical results related to these variables can be found in Chapter 6 of Zipkin [2000]. Appendix E provides some basic analytical formulas

related to an  $(r, Q)$  inventory model with Poisson demand. Let us take a look at an example that illustrates an inventory situation.

### EXAMPLE 8.10 $(r, Q)$ Inventory Policy Evaluation

An inventory manager is interested in understanding the cost and service trade-offs related to the inventory management of computer printers. Suppose customer demand occurs according to a Poisson process at a rate of 3.6 units per month and the lead-time is 0.5 months. The manager has estimated that the holding cost for the item is approximately \$0.25 per unit per month. In addition, when a backorder occurs, the estimate cost will be \$1.75 per unit per month. Every time that an order is placed, it costs approximately \$0.15 to prepare and process the order. The inventory manager has set the reorder point to 1 units and the reorder quantity to 2 units. Develop a simulation model that estimates the following quantities:

- (a) Average inventory on hand and backordered
- (b) Average frequency at which orders are placed
- (c) Probability that an arriving customer does not have their demand immediately filled
- (d) Average total cost per time.

*Solution to Example 8.10:* The Arena™ model will follow closely the pseudo-code outlined in Exhibit 8.6. To develop this model, first define the variables to be used as per Figure 8.61. The reorder point and reorder quantity have been set to ( $r = 1$ ) and ( $Q = 2$ ), respectively. The costs have been set based on the example. Notice that the report statistics check boxes have been checked for the on-hand inventory and the backordered inventory. This will cause time-based statistics to be collected on these variables.

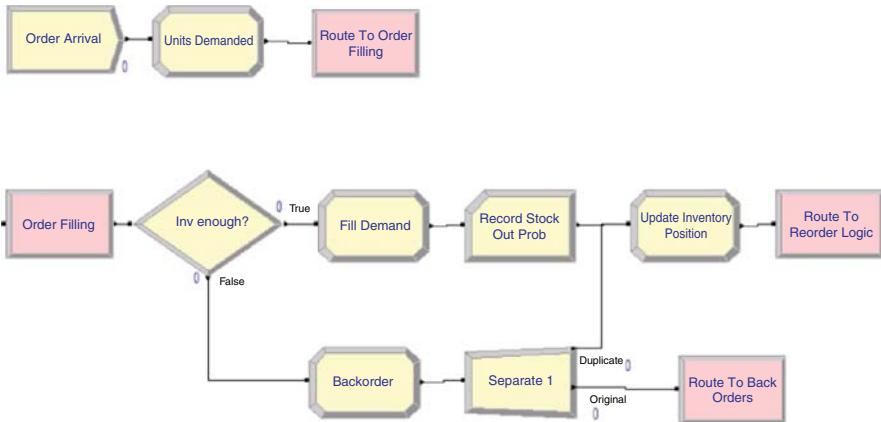
Figure 8.62 illustrates the logic for creating incoming demands and for order fulfillment. First, an entity is created to represent the demand. This occurs according to a time between arrivals with an exponential distribution with a mean of  $(1/0.12)$  days ( $3.6 \text{ units/month} * (1 \text{ month}/30 \text{ days}) = 0.12 \text{ units/day}$ ).

The ASSIGN module in Figure 8.63 shows the amount demanded set to 1 and a stock out indicator set to 0. This will be used to tally the probability that an incoming demand

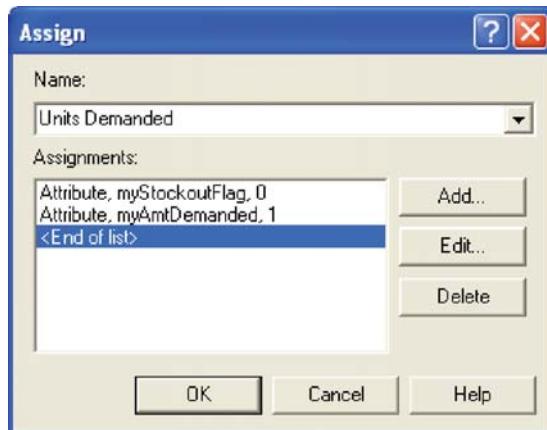
	Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics
1	vReorderPt			Real	System	1 rows	<input type="checkbox"/>
2	vReorderQty			Real	System	1 rows	<input type="checkbox"/>
3	vOnHand			Real	System	0 rows	<input checked="" type="checkbox"/>
4	vInvPos			Real	System	0 rows	<input type="checkbox"/>
5	vOnOrder			Real	System	0 rows	<input type="checkbox"/>
6	vBackOrdered			Real	System	0 rows	<input checked="" type="checkbox"/>
7	vNumOrders			Real	System	0 rows	<input type="checkbox"/>
8	vHoldingCost			Real	System	1 rows	<input type="checkbox"/>
9	vBackOrderCost			Real	System	1 rows	<input type="checkbox"/>
10	vOrderingCost			Real	System	1 rows	<input type="checkbox"/>

Double-click here to add a new row.

**Figure 8.61** Inventory model variables.



**Figure 8.62** Initial filling logic for  $(r, Q)$  inventory model.



**Figure 8.63** Assigning the amount of demand.

is not filled immediately from on-hand inventory. This is called the probability of stocking out. This is the complement of the fill rate. Then, a ROUTE module with zero transfer delay is used to send the demand to a station to handle the order fulfillment. Note that a STATION was used here to represent a logical label to which an entity can be sent.

At the order fulfillment station, the DECIDE module is used to check if the amount demanded is less than or equal to the inventory on hand. If true, the demand is filled using the ASSIGN module to decrement the inventory on hand. The RECORD module is used to collect statistics for the probability of stock out. The inventory position is updated and then the demand is sent to the reordering logic. If false, the demand is backordered. The ASSIGN module labeled Backorder increments the number of backorders. Then the SEPARATE module creates a duplicate, which goes to the station that handles backorders. The original is sent to update the inventory position before being sent to the reordering logic via a ROUTE module.

The Fill Demand, BackOrder, and Update Inventory Position ASSIGN modules have the form shown in Figure 8.64. The amount on hand or the amount backordered is decreased

or increased accordingly. In addition, the inventory position is updated. In the backordering ASSIGN module, the stock out flag is set to 1 to indicate that this particular demand did not get an immediate fill.

When the demand is ultimately filled, it will pass through the RECORD module within Figure 8.62. Notice that in the RECORD module, the expression option is used to record on the attribute, which has a value of 1 if the demand had been backordered upon arrival and the value 0 if it had not been backordered upon initial arrival (Figure 8.65). This indicator variable will estimate the probability that an arriving customer will be backordered. In inventory analysis parlance, this is called the probability of stock out. One minus the probability of stock out is called the fill rate for the inventory system. Under Poisson arrivals, it can be shown that the probability of an arriving customer being backordered will be the same as  $\bar{SO}$ , the percentage of time that the system is stocked out. This is due to a phenomenon called PASTA (Poisson Arrivals See Time Averages) and is discussed in Zipkin [2000] as well as many texts on stochastic processes (e.g., Tijms: [2003]).

The logic representing the backordering and replenishment ordering process is given in Figure 8.66. Notice how STATION modules have been used to denote the start of the logic. In the case of backordering, the Handle Back Order station has a HOLD module, which



Figure 8.64 ASSIGN Modules for filling, backordering, and updating inventory position.

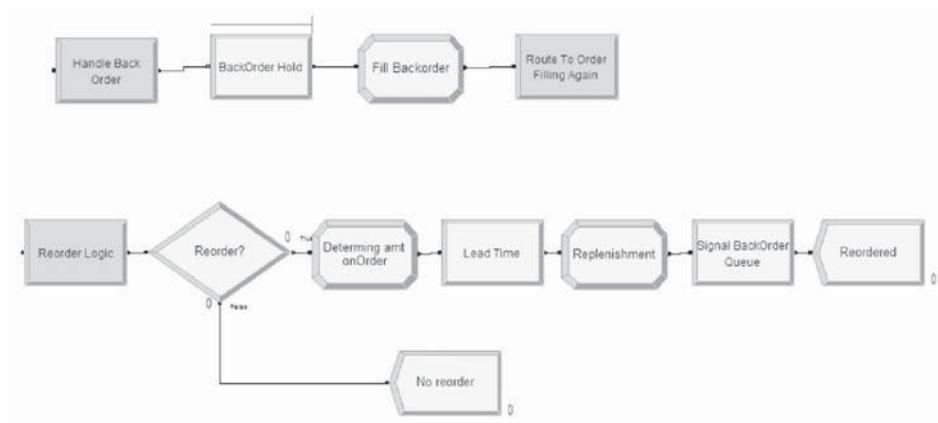


Figure 8.65 Recording demands initial fill condition.

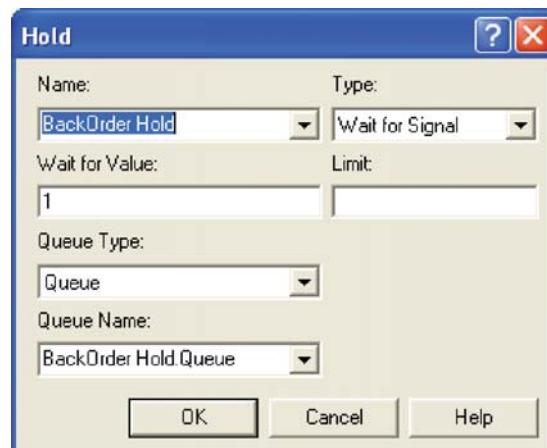
will hold the demands waiting to be filled in a queue. When the HOLD module is signaled (value = 1), the waiting demands are released. The release will be triggered after a replenishment order arrives (Figure 8.67). The number of waiting backorders is decremented in the ASSIGN module and the demands are sent via a ROUTE module for order fulfillment.

At the Reorder Logic station, the DECIDE module checks if the inventory position is equal to or below the reorder point. If this is true, then the amount on order is updated. After the amount of the order is determined, the order entity experiences the delay representing the lead-time. After the lead-time, the order has essentially arrived to replenish the inventory. The corresponding ordering and replenishment ASSIGN modules and the subsequent SIGNAL modules are given in Figures 8.68 and 8.69, respectively.

The basic model structure is now completed; however, there are a few issues related to the collection of the performance measures that must be discussed. The average order frequency must be collected. Recall that  $\overline{OF} = N(T)/T$ . Thus, the number of orders placed within the time interval of interest must be observed. This can be done by creating a logic entity that



**Figure 8.66** Backordering and replenishment logic.



**Figure 8.67** Backorder queue as a HOLD module.

will observe the number of orders that have been placed since the start of the observation period. In Figure 8.68, there is a variable called, *vNumOrders*, which is incremented each time an order is placed. This is, in essence,  $N(T)$ . Figure 8.70 shows the order frequency collection logic. First, an entity is created every,  $T$ , time units. In this case, the interval is monthly (every 30 days).

Then the RECORD module observes the value of *vNumOrders*. The following ASSIGN module sets *vNumOrders* equal to zero. Thus, *vNumOrders* represents the number of orders accumulated during the 30-day period.

To close out the statistical collection of the performance measures, the collection of  $\overline{SO}$  as well as the cost of the policy needs to be discussed. To collect  $\overline{SO}$ , a time-persistent statistic needs to be defined using the Statistic module of the Advanced Process panel as in Figure 8.71. The expression (*vOnHand* == 0) is a Boolean expression, which evaluates to 1.0 if true and 0.0 if false. By creating a time-persistent statistic on this expression, the proportion of time that the on hand is equal to 0.0 can be tabulated.

The figure consists of two separate dialog boxes, both titled "Assignments".

**Top Dialog (Ordering ASSIGN Module):**

	Type	Variable Name	New Value
1	Variable	vOnOrder	vOnOrder + vReorderQty
2	Variable	vNumOrders	vNumOrders + 1

**Bottom Dialog (Replenishment ASSIGN Module):**

	Type	Variable Name	New Value
1	Variable	vOnOrder	vOnOrder - vReorderQty
2	Variable	vOnHand	vOnHand + vReorderQty

Figure 8.68 Ordering and replenishment ASSIGN modules.

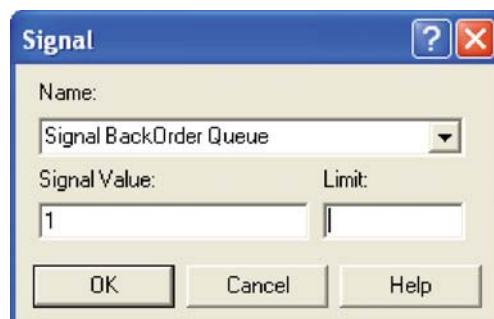


Figure 8.69 Signaling the backorder queue.

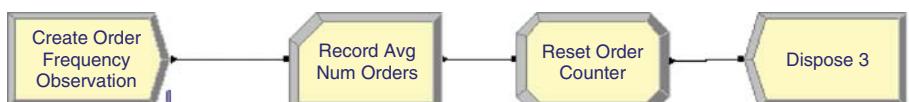


Figure 8.70 Order frequency collection logic.

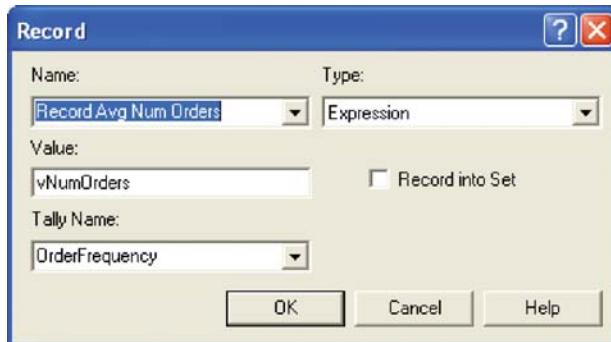


Figure 8.71 Recording the number of orders placed.

	Name	Type	Expression	Report Label	Output File
1	TimeOutOfStock	Time-Persistent	(vOnHand == 0)	TimeOutOfStock	
2	HoldingCost	Output	vHoldingCost * DAVG(vOnHand Value)	HoldingCost	
3	BackorderCost	Output	vBackOrderCost * DAVG(vBackOrdered Value)	BackorderCost	
4	OrderingCost	Output	vOrderingCost * TAVG(OrderFrequency)	OrderingCost	
5	TotalCost	Output	OVALUE(BackorderCost) + OVALUE(HoldingCost) + OVALUE(OrderingCost)	TotalCost	

Double-click here to add a new row.

Figure 8.72 Statistic module for  $(r, Q)$  inventory model.

To record the costs associated with the policy, three Output statistics are needed. Recall that these are designed to be observed at the end of each replication. For the inventory holding cost, the time average of the on-hand inventory variable,  $DAVG(vOnHand Value)$ , should be multiplied by the holding cost per unit per time. The backordering cost is done in a similar manner. The ordering cost is computed by taking the cost per order times the average order frequency. Recall that this was captured via a RECORD module every month. The average from the RECORD module is available through the  $TAVG()$  function. Finally, the total cost is tabulated as the sum of the backordering cost, the holding cost, and the ordering cost. In Figure 8.72, this was accomplished by using the  $OVALUE()$  function for OUTPUT statistics. This function returns the last observed value of the OUTPUT statistic. In this case, it will return the value observed at the end of the replication. Each of these statistics will be shown on the Arena™ reports as user-defined statistics.

The case of  $(r = 1, Q = 2)$  was run for 30 replications have a warm-up period of 3600 days and a run length of 39,600 days. Figure 8.73 indicates that the total cost is about 1.03 per month with a probability of stocking out close to 40%. Notice that the stock out probability is essentially the same as the percentage of time the system was out of stock. This is an indication that the PASTA property of Poison arrivals is working according to theory.

This example should give you a solid basis for developing more sophisticated inventory models. While analytical results were available for the example, small changes in the assumptions necessitate the need for simulation. For example, what if the lead-times are stochastic or the demand is not in units of 1. In the latter case, the filling of the backorder queue should be done in different ways. For example, suppose customers wanting five and three items, respectively, were waiting in the queue. Now suppose a replenishment of four items comes in. Do you give 4 units of the item to the customer wanting 5 units (partial fill)

Expression	Average	Half Width
OrderFrequency	1.8022	0.01
StockOutProb	0.4034	0.01
<b>Time Persistent</b>		
Time Persistent	Average	Half Width
TimeOutOfStock	0.4034	0.00
<b>Variable</b>		
Variable	Average	Half Width
vBackOrdered	0.2952	0.01
vOnHand	0.9926	0.01
<b>Output</b>		
Output	Average	Half Width
BackorderCost	0.5166	0.01
HoldingCost	0.2481	0.00
OrderingCost	0.2703	0.00
TotalCost	1.0351	0.01

**Figure 8.73** Results for the  $(r, Q)$  inventory model.

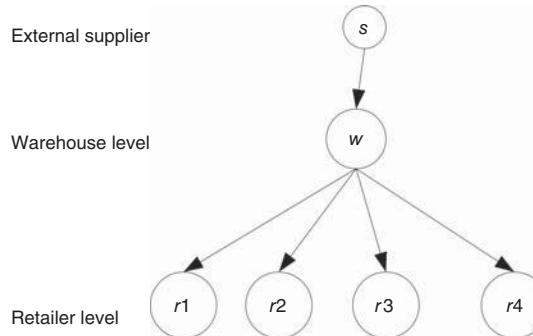
or do you choose the customer wanting 3 units and fill their entire backorder and give only 1 unit to the customer wanting 5 units. The more realism needed, the less analytical models can be applied, and the more simulation becomes useful.

In the next section, the previous inventory system is expanded to handle a supply chain. Because of the complexity, only an overview of the modeling issues will be described. In addition, an Arena™ model capable of modeling a system with a supplier, a warehouse, and multiple retailers having only one product will be presented.

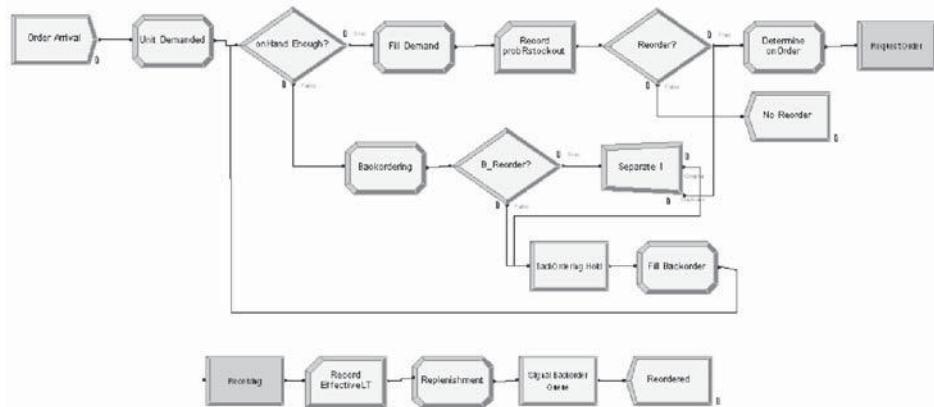
### 8.8.2 Modeling a Multi-Echelon Inventory System

This section describes how to model a simple multi-echelon inventory system. Figure 8.74 illustrates the basic system that will be modeled. The system consists of an external supplier, a warehouse, and a set of retailers. Assume that there is only one item type stocked at each location within the system. The warehouse supplies each of the four retailers when they make a replenishment request. In this model, each of the four retailers will be identical. They have the same control policy settings for their items and they experience the same customer demand. The arrangement of suppliers and customers in this manner results in a tree structure as illustrated in the figure.

In the model, a reorder point reorder quantity  $(r, Q)$  inventory policy is utilized at each location. If the location does not have sufficient stock to satisfy the demand, then the order gets backordered. The retail level experiences customer demand according to a Poisson process. The warehouse experiences replenishment requests for the order quantities,  $Q$ , associated with the retailers. The time between the placement of a replenishment order by a retailer and the arrival of the replenishment from the warehouse is called the lead-time.



**Figure 8.74** A simple multi-echelon inventory system.



**Figure 8.75** Retailer logic for multi-echelon inventory system.

The lead-time can consist of the waiting time to fill the order if backordered plus a transport time to move the order from the warehouse to the retailer. This example assumes that the transport time is deterministic. The warehouse, in turn, will order its replenishment from the external supplier. The external supplier can satisfy any order placed on it, with the order being satisfied after a corresponding delay for the lead-time. Conceptually, the external supplier's lead-time is the production and transport time for the order.

In order to develop an Arena™ model for this situation, the previous example can be expanded. Notice that each location follows the  $(r, Q)$  inventory policy. Thus, the basic logic can be repeated for each retailer and for the warehouse. Then, the retailers must send their orders to the warehouse and the warehouse must send its filled orders to the appropriate retailer. Rather than cutting and pasting the basic logic five times (once for the warehouse and four times for the retailers), the model can be made more generic by using arrays. This will enable the expansion of the model to more retailers if required in the future.

Figure 8.75 shows the logic for the retailer level within the multi-echelon inventory system. As you can see, this logic is essentially the same as that used in the previous example. Because of this similarity, all the details of the model are not presented. A summary of the changes will be emphasized.

	Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics
1	vNumR			Real	System	1 rows	<input type="checkbox"/>
2	vMTBA			Real	System	1 rows	<input type="checkbox"/>
3	vRetailerLT			Real	System	1 rows	<input type="checkbox"/>
4	vWhsLT			Real	System	1 rows	<input type="checkbox"/>
5	vQw			Real	System	1 rows	<input type="checkbox"/>
6	vRw			Real	System	1 rows	<input type="checkbox"/>
7	vQr			Real	System	1 rows	<input type="checkbox"/>
8	vRr			Real	System	1 rows	<input type="checkbox"/>
9	vOnHand	4		Real	System	1 rows	<input type="checkbox"/>
10	vOnOrder	4		Real	System	0 rows	<input type="checkbox"/>
11	vBackOrdered	4		Real	System	0 rows	<input type="checkbox"/>
12	vInvPos	4		Real	System	0 rows	<input type="checkbox"/>
13	vSignalValue	4		Real	System	4 rows	<input type="checkbox"/>
14	vWhsOnHand			Real	System	1 rows	<input checked="" type="checkbox"/>
15	vWhsOnOrder			Real	System	0 rows	<input type="checkbox"/>
16	vWhsBackOrdered			Real	System	0 rows	<input checked="" type="checkbox"/>
17	vWhsInvPos			Real	System	0 rows	<input type="checkbox"/>

Double-click here to add a new row.

**Figure 8.76** Variable module.

In Figure 8.76, the variables are defined for the model. The key difference is the representation of the on hand, on order, backordered, and inventory position for each retailer using Arena™ arrays.

- *vNumR*. This variable indicates the total number of retailers. It is used in the Order Arrival of the CREATE Module to determine the aggregate arrival rate for all retailers (i.e., EXPO(*vMTBA/vNumR*)).
- *vMTBA*. This variable is the mean time between the arrivals of customer demand. If the demand rate is  $\lambda$  per period, the mean interarrival time is  $1/\lambda$ .
- *vRetailerLT*. This variable represents the transport lead-time for the retailer from the warehouse. In the model, it is a constant set at 1 day.
- *vWhsLT*. This variable represents the replenishment lead-time for the warehouse from the supplier. It is assumed to be constant with the value of 1 day.
- *vQw*. This variable is the replenishment reorder quantity for the warehouse.
- *vRw*. This variable is the reorder point at the warehouse.
- *vQr*. This variable is the replenishment reorder quantity for the retailer.
- *vRr*. This variable is the reorder point at the retailers.
- *vOnHand(vNumR)*, *vOnOrder(vNumR)*, *vBackOrdered(vNumR)*, *vInvPos(vNumR)*. These variable arrays represent the amount on hand, on order, backordered, and the inventory position at each retailer.
- *vSignalValue(vNumR)*. This variable array represents the signal value used at each retailer to release the orders that are held in the backorder queue.
- *vWhsOnHand*, *vWhsOnOrder*, *vWhsBackOrdered*, *vWhsInvPos*. These variables represent the amount of inventory on hand, on order, backordered, and the inventory position at the warehouse.

If each retailer experiences Poisson demand at rate  $\lambda_i$ , then the overall demand to the system  $i \lambda = \sum_{i=1}^4 \lambda_i$ . This overall demand rate can be randomly split so that each retail

Assignments			
	Type	Attribute Name	
1	Attribute	myRetailerNum	ANINT(UNIF(0,1)*vNumR+0.5)
2	Attribute	myAmtDemanded	1
3	Attribute	myRetailerSOFlag	0
Double-click here to add a new row.			

Figure 8.77 Assigning the retailer number.

Assignments			
	Type	Other	New Value
1	Other	vOnHand(myRetailerNum)	vOnHand(myRetailerNum) - myAmtDemanded
2	Other	vInvPos(myRetailerNum)	vOnHand(myRetailerNum) + vOnOrder(myRetailerNum) - vBackOrdered(myRetailerNum)
Double-click here to add a new row.			

Figure 8.78 Assignments for filling demand.

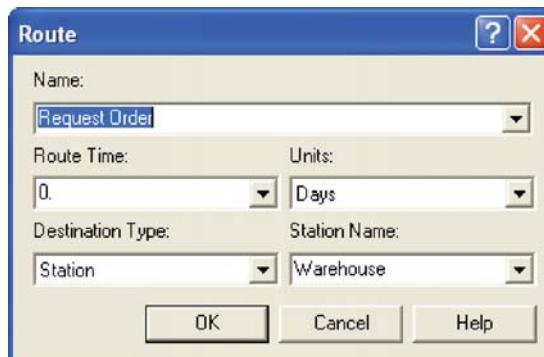


Figure 8.79 Sending the order to the warehouse.

experiences a Poisson process with rate  $\lambda_i$ . See Chapter 1 of Tijms [2003] for more on the merging and splitting properties of the Poisson process. Let  $p_i = \lambda_i/\lambda$  be the probability that an arriving demand should be sent to the  $i$ th retailer. Thus, an arriving demand can be created and then the retailer can be determined according to  $p_i = \lambda_i/\lambda$ . Figure 8.77 illustrates the ASSIGN module after creating the arriving demand.

Since each retailer is identical in the example, the first line randomly generates a discrete uniform number over the range from 1 to 4. After the retailer number has been assigned, this attribute will be used as the index into the arrays representing the retailer's on hand, on order, amount backordered, and inventory position. Figure 8.78 shows an example of how the arrays are used within the ASSIGN module for filling demand. This makes it possible to have any number of retail locations by simply increasing the size of the arrays and appropriately assigning to the attribute *myRetailerNum*.

The rest of the logic for modeling the retail level is the same as described for the previous example, except for the two pink ROUTE and STATION modules. When a retailer decides to place a replenishment order, the entity representing the order is sent via the ROUTE module to a STATION module that denotes the warehouse logic as per Figure 8.79.

When the warehouse fills a demand from a retailer, it is sent to the STATION module labeled Receiving. The outline of the logic associated with the receiving station is as follows:

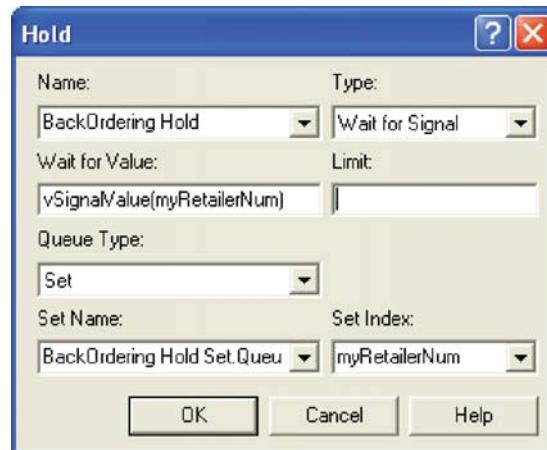
- Receive replenishment Order from warehouse
- Record the effective lead-time Replenishment assignment
  - Decrement the  $vOnOrder(myRetailerNum)$  by  $vQr$
  - Increment the inventory  $vOnhand(myRetailerNum)$  by  $vQr$
- Send signal,  $vSignalValue(myRetailerNum)$ , to release backlog hold queue
- Dispose entity.

The signal that is sent is based on an array indexed by the retailer number. The HOLD module has the ability to hold entities until they receive a specific signal and then to release the entities associated with that signal number. Figure 8.80 shows the corresponding HOLD module associated with the signal. Notice that the Set option has been used for the type of queuing. This allows a separate backorder queue to be defined for each retailer. The set contains four queues, one for each retailer, and it is indexed by the attribute  $myRetailerNum$ . Thus, each backordered demand waits in a queue specific to the retailer that is currently processing the demand.

The logic associated with the warehouse is essentially the same as previously described. The key difference is that the demand sent to the warehouse will require  $vQr$  units of inventory from the warehouse (no longer a single unit). Assume that backordered demands are satisfied on a FIFO basis. Once a demand has been filled, it must be sent back to the proper retailer. This is accomplished by the ROUTE module denoted as *Ship Order* in Figure 8.81.

The ROUTE module is shown in Figure 8.82. The entity is sent directly to the station with the name *Receiving* using a routing time of 1 day ( $vRetailerLT$ ).

Collecting statistics on this model is a bit more tedious. Figure 8.83 illustrates the primary problem. Because the state variables, for example, on hand, are represented as arrays, time-persistent statistics on each element of the array cannot be directly defined in the VARIABLE module. Thus, in order to collect statistics on the average inventory and average amount backordered, a specific time-persistent statistic must be defined as shown in



**Figure 8.80** HOLD module for queuing backorders at the retailers.

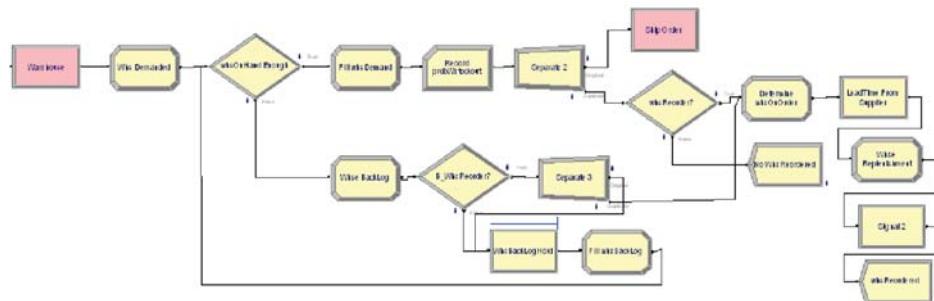


Figure 8.81 Warehouse demand processing logic.

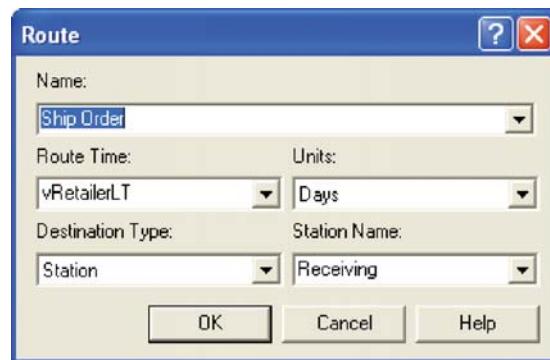


Figure 8.82 Shipping the order to the retailer.

Statistic - Advanced Process					
	Name	Type	Expression	Report Label	Output File
1	onHandR1	Time-Persistent	vOnHand(1)	onHandR1	
2	onHandR2	Time-Persistent	vOnHand(2)	onHandR2	
3	backOrderR1	Time-Persistent	vBackOrdered(1)	backOrderR1	
4	backOrderR2	Time-Persistent	vBackOrdered(2)	backOrderR2	
5	backOrderR3	Time-Persistent	vBackOrdered(3)	backOrderR3	
6	backOrderR4	Time-Persistent	vBackOrdered(4)	backOrderR4	
7	onHandR3	Time-Persistent	vOnHand(3)	onHandR3	
8	onHandR4	Time-Persistent	vOnHand(4)	onHandR4	

Double-click here to add a new row.

Figure 8.83 Statistic module for multi-echelon inventory model.

Figure 8.83. Other statistics need to be handled in a similar manner. These extensions are left as exercises.

This completes the discussion of the multi-echelon inventory system. To run the model, you must set the reorder point and reorder quantity for the warehouse and retailers. As an example, the model was run with the settings shown in Table 8.5.

<b>Tally</b>		
Expression	Average	Half Width
Record probRstockout	0.1223	0.01
Record probWstockout	0.3488	0.01
Interval		
Record EffectiveLT	Average	Half Width
	1.3006	0.00
<b>Time Persistent</b>		
Time Persistent	Average	Half Width
backOrderR1	0.00893928	0.00
backOrderR2	0.00875987	0.00
backOrderR3	0.00974924	0.00
backOrderR4	0.00872368	0.00
onHandR1	0.8766	0.01
onHandR2	0.8782	0.00
onHandR3	0.8743	0.01
onHandR4	0.8800	0.00
Variable		
Variable	Average	Half Width
vWhsBackOrdered	0.1219	0.01
vWhsOnHand	1.2164	0.02

**Figure 8.84** Results for multi-echelon inventory system.

Figure 8.84 shows the results for running the multi-echelon inventory system. The average inventory on hand and on backorder for each of the retailers is shown. As an extension, one might want to collect statistics at the aggregate level (e.g., average amount of inventory in the system or at the retailer level).

**TABLE 8.5 Parameters for Running the Multi-Echelon Inventory System**

Parameter	Variable Name	Value
Number of retailers	vNumR	4
Mean time between demand arrivals	vMTBA	10 days
Retailer lead-time	vRetailerLT	1 day
Warehouse lead-time	vWhsLT	1 day
Warehouse reorder quantity	vQw	4
Warehouse reorder point	vRw	-1
Retailer reorder point	vQr	1
Retailer reorder point	vRr	0
Number of replications		10
Warm-up period		360 days
Replication length		3600 days

This is one of the more complicated models presented within the text; however, it should be evident that extending this model to have such elements as shipping via trucks or to have multiple items at each location should be easy to accomplish. The reader is asked to explore some of these extensions in the exercises.

## 8.9 SUMMARY

This chapter provided an introduction to two very important application areas for simulation: queuing and inventory models. For queuing models, you learned how to analyze situations involving a single station using analytical formulas. This analysis involves the following:

1. Identifying the arrival and service processes
2. Identifying the size of the arriving population and the size of the system
3. Specifying the appropriate queuing model and its input parameters
4. Identifying the desired performance measures
5. Computing the required performance measures.

Two key concepts were also covered: Little's Formula and steady-state analysis. Little's formula represents a conservation principle that relates the average waiting time in the system to the average number in the system. The steady-state performance of the queuing system is readily available based on the steady-state probability distribution for the number of customers in the system. This distribution can be thought of as the probability of having a given number of customers in the system at some arbitrary point in the infinite future. Under certain conditions, analytical formulas are available; however, when they are not available, simulation can be used.

Even for simple single station models, small changes to assumptions or the structure of the system necessitate the use of simulation. For example, if the priority of the customer depends on the amount of resource available, then simulation is needed. Queuing systems often involve a network of stations. While analytical model exist for certain classes of system structure, simulation quickly becomes the most viable method for analysis. Arena™ has important constructs that facilitate the modeling of queuing networks:

**STATION Module.** Allows the marking in the model for a location to which entities can be directed for processing.

**SEQUENCE Module.** Allows for prespecified routes of stations to be defined and attributes to be assigned when entities are transferred to the stations.

**ROUTE Module.** Facilitates the movement between stations with a time delay.

In addition to queuing models, you learned about inventory models and the basic operation of the  $(r, Q)$  inventory policy. Again, the application of analytical models quickly becomes limited. When this occurs, simulation provides a robust and useful tool for analyzing systems involving these elements. In particular, a supply chain can be conceptualized as a network of inventory locations. Again, the modeling of these kinds of systems can be modeled using the STATION and ROUTE modules. The analysis of these types of system

has only been at an introductory level. You should refer to a number of the excellent texts mentioned in the references to learn more about these topics.

The modeling journey is not yet complete. So far, you should have developed a good working knowledge on how to utilize Arena<sup>TM</sup> to model a number of interesting situations. The next chapter takes this modeling even further by studying constructs within Arena<sup>TM</sup> that facilitate the modeling of systems involving material movement. These constructs include transporters, conveyors, and automated guided vehicles.

## EXERCISES

For Exercises (8.4)–(8.22), specify the appropriate queuing models needed to solve the exercise using Kendall's notation. Specify the parameters of the model, for example,  $\lambda_e$ ,  $\mu$ ,  $c$ , size of the population, and size of the system. Specify how and what you would compute to solve the problem. Be as specific as possible by specifying the equations needed. Then, compute the desired quantities. You might also try to use Arena<sup>TM</sup> to solve the problems via simulation.

- 8.1 *True or False:* In a queuing system with random arrivals and random service times, the performance will be best if the arrival rate is equal to the service rate because then there will not be any queuing.
- 8.2 In modeling reneging from a queue, a (a)\_\_\_\_\_ module was used to schedule the time to renege and a (b)\_\_\_\_\_ module was used to find the entity's duplicate in the queue.
- 8.3 The BK in the UA food court uses an average of 10,000 pounds of potatoes per week. The average number of pounds of potatoes on hand is 5000. On an average, how long do potatoes stay in the restaurant before being used? What queuing concept is this question an application of?
- 8.4 Consider a single pump gas station where the arrival process is Poisson with a mean time between arrivals of 10 minutes. The service time is exponentially distributed with a mean of 6 minutes. Specify the appropriate queuing model needed to solve the problem using Kendall's notation. Specify the parameters of the model and what you would compute to solve the problem. Be as specific as possible by specifying the equation needed. Then, compute the desired quantities.
  - (a) What is the probability that you have to wait for service?
  - (b) What is the mean number of customer at the station?
  - (c) What is the expected time waiting in the line to get a pump?
- 8.5 Suppose an operator has been assigned to the responsibility of maintaining three machines. For each machine, the probability distribution of the running time before a breakdown is exponentially distributed with a mean of 9 hours. The repair time also has an exponential distribution with a mean of 2 hours. Specify the appropriate queuing model needed to solve the problem using Kendall's notation. Specify the parameters of the model and what you would compute to solve the problem. Be as specific as possible by specifying the equation needed. Then, compute the desired quantities.
  - (a) What is the probability that the operator is idle?

- (b) What is the expected number of machines that are running?  
(c) What is the expected number of machines that are not running?
- 8.6 SuperFastCopy wants to install self-service copiers but cannot decide whether to put in one or two machines. They predict that arrivals will be Poisson with a rate of 30 per hour, and the time spent copying is exponentially distributed with a mean of 1.75 minutes. Because the shop is small, they want the probability of 5 or more customers in the shop to be small, say less than 7%. Make a recommendation based on queuing theory to SuperFastCopy.
- 8.7 Each airline passenger and his or her carry-on baggage must be checked at the security checkpoint. Suppose XNA averages 10 passengers per minute with exponential interarrival times. To screen passengers, the airport must have a metal detector and baggage X-ray machines. Whenever a checkpoint is in operation, two employees are required (one operates the metal detector and one operates the X-ray machine). The passenger goes through the metal detector and simultaneously their bag goes through the X-ray machine. A checkpoint can check an average of 12 passengers per minute according to an exponential distribution.
- (a) What is the probability that a passenger will have to wait before being screened?  
(b) On an average, how many passengers are waiting in line to enter the checkpoint?  
(c) On an average, how long will a passenger spend at the checkpoint?
- 8.8 Two machines are being considered for processing a job within a factory. The first machine has an exponentially distributed processing time with a mean of 10 minutes. For the second machine, the vendor has indicated that the mean processing time is 10 minutes but with a standard deviation of 6 minutes. Using queuing theory, which machine is better in terms of the average waiting time of the jobs?
- 8.9 Customers arrive at a one-window drive in bank according to a Poisson distribution with a mean of 10 per hour. The service time for each customer is exponentially distributed with a mean of 5 minutes. There are three spaces in front of the window including that for the car being served. Other arriving cars can wait outside these three spaces. Specify the appropriate queuing model needed to solve the problem using Kendall's notation. Specify the parameters of the model and what you would compute to solve the problem. Be as specific as possible by specifying the equation needed. Then, compute the desired quantities.
- (a) What is the probability that an arriving customer can enter one of the three spaces in front of the window?  
(b) What is the probability that an arriving customer will have to wait outside the three spaces?  
(c) How long is an arriving customer expected to wait before starting service?  
(d) How many spaces should be provided in front of the window so that an arriving customer can wait in front of the window at least 20% of the time? In other words, the probability of at least one open space must be greater than 20%.
- 8.10 Joe Rose is a student at Big State U. He does odd jobs to supplement his income. Job requests come every 5 days on an average, but the time between requests is exponentially distributed. The time for completing a job is also exponentially distributed with a mean of 4 days.

- (a) What would you compute to find the chance that Joe will not have any jobs to work on?
  - (b) What would you compute to find the average value of the waiting jobs if Joe gets about \$25 per job?
- 8.11 The manager of a bank must determine how many tellers should be available. For every minute a customer stands in line, the manager believes that a delay cost of 5 cents is incurred. An average of 15 customers per hour arrive at the bank. On an average, it takes a teller 6 minutes to complete the customer's transaction. It costs the bank \$9 per hour to have a teller available. Interarrival and service times can be assumed to be exponentially distributed.
- (a) What is the minimum number of tellers that should be available in order for the system to be stable (i.e., not have an infinite queue)?
  - (b) If the system has three tellers, what is the probability that there will be no one in the bank?
  - (c) What is the expected total cost of the system per hour, when there are two tellers?
- 8.12 You have been hired to analyze the needs for loading dock facilities at a trucking terminal. The present terminal has four docks on the main building. Any trucks that arrive when all docks are full are assigned to a secondary terminal, which is a short distance away from the main terminal. Assume that the arrival process is Poisson with a rate of five trucks each hour. There is no available space at the main terminal for trucks to wait for a dock. At the present time, nearly 50% of the arriving trucks are diverted to the secondary terminal. The average service time per truck is 2 hours on the main terminal and 3 hours on the secondary terminal, both exponentially distributed. Two proposals are being considered. The first proposal is to expand the main terminal by adding docks so that at least 80% of the arriving trucks can be served there with the remainder being diverted to the secondary terminal. The second proposal is to expand the space that can accommodate up to eight trucks. Then, only when the holding area is full, the trucks will be diverted to secondary terminal.
- (a) What queuing model should you use to analyze the first proposal? State the model and its parameters.
  - (b) State what you would do to determine the required number of docks so that at least 80% of the arriving trucks can be served for the first proposal. Note you do not have to compute anything.
  - (c) What model should you use to analyze the second proposal? State the model and its parameters.
- 8.13 Sly's convenience store operates a two-pump gas station. The lane leading to the pumps can house at most five cars, including those being serviced. Arriving cars go elsewhere if the lane is full. The distribution of the arriving cars is Poisson with a mean of 20 per hour. The time to fill up and pay for the purchase is exponentially distributed with a mean of 6 minutes.
- (a) Specify using queuing notation, exactly what you would compute to find the percentage of cars that will seek business elsewhere?
  - (b) Specify using queuing notation, exactly what you would compute to find the utilization of the pumps?

- 8.14 An airline ticket office has two ticket agents answering incoming phone calls for flight reservations. In addition, two callers can be put on hold until one of the agents is available to take the call. If all four phone lines (both agent lines and the hold lines) are busy, a potential customer gets a busy signal, and it is assumed that the call goes to another ticket office and that the business is lost. The calls and attempted calls occur randomly (i.e., according to Poisson process) at a mean rate of 15 per hour. The length of a telephone conversation has an exponential distribution with a mean of 4 minutes.
- Specify using queuing notation, exactly what you would compute to find the probability of losing a potential customer?
  - What would you compute to find the probability that an arriving phone call will not start service immediately but will be able to wait on a hold line?
- 8.15 SuperFastCopy has three identical copying machines. When a machine is being used, the time until it breaks down has an exponential distribution with a mean of 2 weeks. A repair person is kept on call to repair the machines. The repair time for a machine has an exponential distribution with a mean of 0.5 week. The downtime cost for each copying machine is \$100 per week.
- Let the state of the system be the number of machines not working. Construct a state-transition diagram for this queuing system.
  - Write an expression using queuing performance measures to compute the expected downtime cost per week.
- 8.16 NWH Cardiac Care Unit (CCU) has five beds, which are virtually always occupied by patients who have just undergone major heart surgery. Two registered nurses (RNs) are on duty in the CCU in each of the three 8-hour shifts. About every 2 hours following an exponential distribution, one of the patients requires a nurse's attention. The RN will then spend an average of 30 minutes (exponentially distributed) assisting the patient and updating medical records regarding the problem and care provided. Because immediate service is critical to the five patients, two important questions are
- What would you compute to find the average number of patients being attended by the nurses?
  - What would you compute to find the average time that a patient spends waiting for one of the nurses to arrive?
- 8.17 HJ Bunt, Transport Company maintains a large fleet of refrigerated trailers. For the purposes of this problem, assume that the number of refrigerated trailers is conceptually infinite. The trailers require service on an irregular basis in the company owned and operated service shop. Assume that the arrival of trailers to the shop is approximated by a Poisson distribution with a mean rate of 3 per week. The length of time needed for servicing a trailer varies according to an exponential distribution with a mean service time of one-half week per trailer. The current policy is to utilize a centralized contracted outsourced service center whenever more than two trailers are in the company shop, so that at most one trailer is allowed to wait. Assume that there is currently only one mechanic in the company shop.
- Specify using Kendall's notation the correct queuing model for this situation including the appropriate parameters.

- (b) Compute to determine the expected number of repairs that are outsourced per week?
- 8.18 Rick is a manager of a small barber shop at Big State U. He hires one barber. Rick is also a barber and he works only when he has more than one customer in the shop. Customers arrive randomly at a rate of 3 per hour. Rick takes 15 minutes on an average for a hair cut, but his employee takes 10 minutes. Assume that the cutting time distributions are exponentially distributed. Assume that there are only two chairs available with no waiting room in the shop.
- Let the state of the system be the number of customers in the shop, Construct a state-transition diagram for this queuing system.
  - What is the probability that a customer is turned away?
  - What is the probability that the barber shop is idle?
  - What is the steady-state mean number of customers in the shop?
- 8.19 Using the supplied data set, draw the sample path for the state variable,  $N(t)$ , which represents the number of customers in a queuing system at any time  $t$ . Assume that the value of  $N(t)$  is the value of the state variable just after time  $t$ .

$t$	0	2	4	5	7	10	12	15	20
$N(t)$	0	1	0	1	2	3	2	1	0

- Give a formula for estimating the time-average number in the system,  $N(t)$ , and then use the data to compute the time-average number in the system over the range from 0 to 25.
  - Give a formula for estimating the mean rate of arrivals over the interval from 0 to 25 and then use the data to estimate the mean arrival rate.
  - Estimate the average time in the system (waiting and in service) for the customers indicated in the diagram.
- 8.20 A patient arrives at the Emergency Room about every  $20 \pm 10$  minutes (stream 1). The notation  $X \pm Y$  means uniformly distributed with minimum  $X - Y$  and maximum  $X + Y$ . They will be treated by either of two doctors.

Twenty percent of the patients are classified as NIA (need immediate attention) and the rest as CW (can wait). NIA patients are given the highest priority, 3, see a doctor as soon as possible for  $40 \pm 37$  minutes (stream 2), then have their priority reduced to 2 and wait until a doctor is free again, when they receive further treatment for  $30 \pm 25$  minutes (stream 3) and are discharged.

CW patients initially receive a priority of 1 and are treated (when their turn comes) for  $15 \pm 14$  minutes (stream 4); their priority is then increased to 2, they wait again until a doctor is free, receive  $10 \pm 8$  minutes (stream 5) of final treatment and are discharged.

An important aspect of this system is that patients who have already seen the doctor once compete with newly arriving patients who need a doctor. As indicated, patients who have already seen the doctor once have a priority level of 2 (either increased from 1 to 2 or decreased from 3 to 2). Thus, there is one shared queue for the first treatment activity and the final treatment activity. Hint: While there are a number of

ways to address this issue, you might want to look up Shared Queue in the Arena™ Help files. In addition, we assume that the doctors are interchangeable. That is, it does not matter which of the two doctors performs the first or final treatment. Assume that the initial treatment activity has a higher priority over the final treatment for a doctor.

Simulate for 20 days of continuous operation, 24 hours per day. Precede this by a 2-day initialization period to load the system with patients.

- (a) Measure the average queue length of NIA patients from arrival to first seeing a doctor.
  - (b) What percentage have to wait to see the doctor for the first treatment?
  - (c) Report statistics on the initial waiting time for NIA patients.
  - (d) What percentage wait less than 5 minutes before seeing a doctor for the first time?
  - (e) Report statistics on the time in system for the patients.
  - (f) Report statistics on the remaining time in system from after the first treatment to discharge for all patients.
  - (g) Discuss what changes to your model are necessary if the doctors are not interchangeable. That is, suppose there are two doctors: Dr. House and Dr. Wilson. The patient must get the same doctor for their final treatment as for their first treatment. For example, if a patient gets Dr. House for their first treatment, they must see Dr. House for their final treatment. You do not have to implement the changes.
- 8.21 Consider the M/G/1 queue with the following variation. The server works continuously as long as there is at least one customer in the system. The customers are processed FIFO. When the server finishes serving a customer and finds the system empty, the server goes away for a length of time called a vacation. At the end of the vacation, the server returns and begins to serve the customers, if any, who have arrived during the vacation. If the server finds no customers waiting at the end of a vacation, it immediately takes another vacation and continues in this manner until it finds at least one waiting customer upon return from a vacation. Assume that the time between customer arrivals is exponentially distributed with mean of 3 minutes. The service distribution for each customer is a gamma distribution with a mean of 4.5 seconds and a variance of 3.375. The length of a vacation is a random variable uniformly distributed between 8 and 12 minutes. Run the simulation long enough to adequately develop a 95% confidence interval on the expected wait time in the queue for a arbitrary customer arriving in steady state. In addition, develop an empirical distribution for the number of customers waiting upon the return of the server from vacation. In other words, estimate the probability that  $j = 0, 1, 2, \dots$ , where  $j$  is the number of waiting customers in the queue upon the return of the server from a vacation. This queue has many applications, for example, consider how a bus stop operates.
- 8.22 Reconsider Exercise (8.14). Suppose the airline ticket office has expanded the number of lines to 8 lines, still with two agents. In addition, they have instituted an automated caller identification system that automatically places First Class Business (FCB) passengers at the head of the queue, waiting for an agent. Of the original 15 calls per hour, they estimate that roughly one-third of these come from FCB customers. They have also noticed that FCB customers take approximately 3 minutes on an average for the phone call, still exponentially distributed. Regular customers still take on an

- average 4 minutes, exponentially distributed. Simulate this system with and without the new prioritization scheme and compare the average waiting time for the two types of customers.
- 8.23 Consider a system having six machines tended by two operators. In this system, there are two types of stoppages. Type 1 stoppage occurs after a fixed constant amount of machine running time,  $1/\lambda_1 = 30$  minutes, and has a constant value of  $1/\mu_1 = 10$  minutes as the service time for the stoppage. Type 2 stoppages occur after random intervals of time, negatively exponentially distributed, with a mean of  $1/\lambda_2 = 10$  minutes. Service times for type 2 stoppages are negative exponentially distributed with a mean of  $1/\mu_2 = 4$  minutes. Both of the operators have the same skills and can handle either type of stoppage. The machines wait for service from the operators in a FCFS queue with no priority given to either type of stoppage. Simulate this system for 10,000 minutes to estimate the average number of waiting machines by type of stoppage, the average utilization of the operator, the average utilization of the machines, and the average waiting time of the machines by type of stoppage.
- 8.24 Consider the walk-in health care clinic example. The management of the clinic is interested in improving the handling of low priority patients. To that end, they are considering hiring a special registered nurse practitioner who is licensed to provide minor medical care under the supervision of doctors. In changing the system, the low priority patients will be handled by the nurse practitioner after triage. The balking and reneging will still occur as before, except the patients now wait in a dedicated area for the nurse practitioner and do not wait in the main line for the doctor. Management predicts that 80% of the low priority patients will be able to be handled solely by the nurse practitioner; however, after seeing the nurse practitioner, 20% of the patients must also see the a doctor. The service time at the nurse practitioner is the same as it would have been if the low priority patient had visited the doctor. Those low priority patients who must see the doctor are placed at the head of the line for the doctor queue. When these patients visit the doctor, their service time is lognormally distributed with a mean of 6 minutes and a standard deviation of 1 minute. Compare the operation of the proposed system to the current operation of the clinic based on 30 replications. Do you think that the dedicated nurse practitioner is a good idea?
- 8.25 Passengers arrive at an airline terminal according to an exponential distribution for the time between arrivals with a mean of 1.5 minutes (stream 1). Of the arriving passengers, 7% are frequent flyers (stream 2). The time that it takes the passenger to walk from the main entrance to the check-in counter is uniform between 2.5 and 3.5 minutes (stream 3). Once at the counter, the travellers must wait in a single line until one of the four agents is available to serve them. The check-in time (in minutes) is gamma distributed with a mean of 5 minutes and a standard deviation of 4 minutes (stream 4). When their check-in is completed, passengers with carry-on only go directly to security. Those with a bag to check, walk to another counter to drop their bag. The time to walk to bag check is uniform(1,2) minutes (stream 9). Since the majority of the flyers are business, only 25% of the travellers has a bag to check (stream 5). At the baggage check, there is a single line served by one agent. The time to drop off the bag at the bag check stations is lognormally distributed with a mean of 2 minutes and a standard deviation of 1 minute (stream 6). After dropping off their bags, the traveller goes to security. The time to walk to security (either after bag check or directly from the check in counter) is exponentially distributed with a mean of 8 minutes (stream 10). At the security check point, there is a single line, served by two TSA

agents. The TSA agents check the boarding passes of the passengers. The time that it takes to check the boarding pass is triangularly distributed with parameters (2, 3, 4) minutes (stream 7). After getting their identity checked, the travellers go through the screening process. We are not interested in the screening process. However, the time that it takes to get through screening is distributed according to a triangular distribution with parameters (5, 7, 9) minutes (stream 8). After screening, the walking time to the passenger's gate is exponentially distributed with a mean of 5 minutes (stream 11).

We are interested in estimating the average time it takes from arriving at the terminal until a passenger gets to their gate. This should be measured overall and by type (frequent flyer vs nonfrequent flyer).

Assume that the system should be studied for 16 hours per day.

- (a) Report the average and 95% confidence interval half-width on the following based on the simulation of 10 days of operation. Report all time units in minutes.

Statistic	Average	Half-Width
Utilization of the check-in agents		
Utilization of the TSA agents		
Utilization of the Bag check agent		
Frequent flyer total time to gate		
Nonfrequent flyer total time to gate		
Total time to gate regardless of type		
Number of travellers in the system		
Number of travellers waiting for check-in		
Number of travellers waiting for security		
Time spent waiting for check-in		
Time spent waiting for security		

- (b) Based on the results of part (a), determine the number of replications necessary to estimate the total time to reach their gate regardless of type to within  $\pm 1$  minute with 95% confidence.
- (c) The airport is interested in improving the time spent by the travellers in the system by considering the following three alternatives:
- A** There are two waiting lines at check-in. One line is dedicated to frequent flyers and one line is dedicated to nonfrequent flyers. Assign three of the current check-in agents to serve nonfrequent flyers only and the remaining agent to serve frequent flyers only. Customers must enter the appropriate line. There are two lines at boarding pass review. One line is for frequent flyers and the other is for nonfrequent flyers. One TSA agent is dedicated to nonfrequent flyers. The other TSA agent is shared between the two. The TSA agent that serves both shows higher priority to frequent flyers.
- B** There are two waiting lines. One line is dedicated to frequent flyers and one line is dedicated to nonfrequent flyers. There are three check-in agents to serve nonfrequent flyers and the fourth agent can serve both frequent flyers and non-frequent flyers. In other words, the agent dedicated to frequent flyers is shared. Nonfrequent-flyer agents should be selected before the frequent-flyer agent

when serving the nonfrequent-flyer line, when more than one agent is available and a customer is waiting. When there are no frequent flyers waiting in the frequent-flyer queue, the agent can serve a nonfrequent flyer if one is waiting. The agents serve the lines with the same priority. There are two lines at boarding pass review. One line is for frequent flyers and the other is for nonfrequent flyers. One TSA agent is dedicated to nonfrequent flyers. The other TSA agent is shared between the two. The TSA agent who serves both shows higher priority to frequent flyers.

- C** This alternative is the same as alternative B, except that the frequent-flyer agent serves frequent flyers with higher priority than waiting nonfrequentflyers at check-in.

Report the average and 95% confidence interval half-width for the following based on the simulation of 50 days of operation:

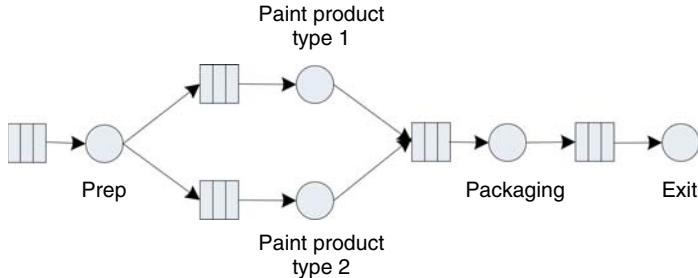
Alternative	Statistic	Average	Half-Width
A	Frequent flyer total time in the system		
A	Nonfrequent flyer total time in the system		
B	Frequent flyer total time in the system		
B	Nonfrequent flyer total time in the system		
C	Frequent flyer total time in the system		
C	Nonfrequent flyer total time in the system		

- (d) Recommend with 95% confidence the best alternative A, B, or C for frequent flyers in terms of total time in the system. Use an indifference parameter of 0 minutes.
- (e) Recommend with 95% confidence the best alternative A, B, or C for nonfrequent flyers in terms of total time in the system. Use an indifference parameter of 0 minutes.
- (f) Animate this model using the ROUTE animation constructs.
- 8.26 Suppose a service facility consists of two stations in series (tandem), each with its own FIFO queue. Each station consists of a queue and a single server. A customer completing service at station 1 proceeds to station 2, while a customer completing service at station 2 leaves the facility. Assume that the interarrival times of customers to station 1 are IID exponential random variables with a mean of 1 minute. Service times of customers at station 1 are exponential random variables with a mean of 0.7 minute, and at station 2 are exponential random variables with mean 0.9 minute. Develop an Arena™ model for this system using the STATION and ROUTE modules.
- (a) Run the simulation for exactly 20,000 minutes and estimate for each station the expected average delay in queue for the customer, the expected time-average number of customers in queue, and the expected utilization. In addition, estimate the average number of customers in the system and the average time spent in the system.
- (b) Use the results of queuing theory to verify and validate your results for part (a)

- (c) Suppose now there is a travel time from the exit of station 1 to the arrival to station 2. Assume that this travel time is distributed uniformly between 0 and 2 minutes. Modify your simulation and rerun it under the same conditions as in part (a).
- 8.27 Reconsider Exercise (8.27). Suppose that there is limited space at the second station. In particular, there is room for one customer to be in service at the second station and room for only one customer to wait at the second station. A customer completing service at the first station will not leave the service area at the first station unless there is a space available for his/her to wait at the second station. In other words, the customer will not release the server at the first station unless he/she can move to the second station. Reanalyze this situation under the same conditions of Exercise (8.27).
- (a) Use a resource to model the waiting space at the second station. Ensure that a customer leaving the first station does not release his/her resource until he/she is able to seize the space at the second station.
  - (b) Use a HOLD module with the wait and signal option to model this situation. Compare your results to those obtained in part (a).
  - (c) Use a HOLD module with the scan for condition option to model this situation. Compare your results to parts (a) and (b).
  - (d) Discuss the effectiveness of each of the methods. Discuss the efficiency of each of the methods.
- 8.28 Assume that we have a single server that performs service according to an exponential distribution with a mean of 0.7 hours. The time between arrival of customers to the server is exponential with mean of 1 hour. If the server is busy, the customer waits in the queue until the server is available. The queue is processed according to a FIFO rule. Use HOLD/SIGNAL to model this situation. Unlike in the second implementation of the M/M/1 in Section 8.6.1, develop your model so that the customer has the delay for service in its process flow.
- 8.29 (*This problem is based on an example on page 209 and continues on page 217 of Pegden et al. [1995]. Used with permission.*) Consider the simple three-workstation flow line. Parts entering the system are placed at a staging area for transfer to the first workstation. The staging area can be thought as the place where the parts enter the system prior to going to the first workstation. No processing takes place at the staging area, other than preparation to be directed to the appropriate stations. After the parts have completed processing at the first workstation, they are transferred to a paint station manned by a second worker, and then to a packaging station where they are packed by a third worker, and then to a second staging area where they exit the system.

The time between part arrivals at the system is exponentially distributed with a mean of 28 minutes (stream 1). The processing time at the first workstation is uniformly distributed between 21 and 25 minutes (stream 2). The paint time is lognormally distributed with a mean of 22 minutes and a standard deviation of 4 (stream 3). The packing time follows a triangular distribution with a minimum of 20, mode of 22, and a maximum of 26 (stream 4). The transfers are unconstrained, in that they do not require a vehicle or resource, but all transfer times are exponential with a mean of 2 or 3 minutes (stream 5). Transfer times from the staging to the workstation and from

pack to exit are 3 minutes. Transfer times from the workstation to paint and from paint to pack are 2 minutes. The performance measures of interest are the utilization and work-in-progress (WIP) at the workstation, paint and packaging operations. A figure of the system is given below.



Suppose that statistics on the part flow time, that is, the total time a part spends in the system need to be collected. However, before simulating this process, it is discovered that a new part needs to be considered. This part will be painted a different color. Because the new part replaces a portion of the sales of the first part, the arrival process remains the same, but 30% of the arriving parts are randomly designated as the new type of part (stream 10). The remaining parts (70% of the total) are produced in the same manner as described above. However, the new part requires the addition of a different station with a painting time that is lognormally distributed with a mean of 49 minutes and a standard deviation of 7 minutes (stream 6). Assume that an additional worker is available at the new station. The existing station paints only the old type of part and the new station paints only the new parts. After the painting operation, the new part is transferred to the existing packing station and requires a packing time that follows a triangular distribution with a minimum value of 21, a mode of 23, and a maximum of 26 (stream 7). Run the model for 600,000 minutes with a 50,000-minute warm-up period. If you were to select a resource to add capacity, which would it be?

- 8.30 Consider the testing and repair shop. Suppose instead of increasing the overall arrival rate of jobs to the system, the new contract will introduce a new type of component into the system that will require a new test plan sequence. The following two tables represent the specifics associated with the new testing plan.

Test Plan	Percentage of Parts (%)	Sequence
1	20	2,3,2,1
2	12.5	3,1
3	37.5	1,3,1
4	20	2,3
5	10	2,1,3

Test Plan	Testing Time Parameters	Repair Time Parameters
1	(20,4.1), (12,4.2), (18,4.3), (16,4.0)	(30,60,80)
2	(12,4), (15,4)	(45,55,70)
3	(18,4.2), (14,4.4), (12,4.3)	(30,40,60)
4	(24,4), (30,4)	(35,65,75)
5	(20,4.1), (15,4), (12,4.2)	(20,30,40)

They are interested in understanding where the potential bottlenecks are in the system and in developing alternatives to mitigate those bottlenecks so that they can still handle the contract. The new contract stipulates that 80% of the time, the testing and repairs should be completed within 480 minutes. The company runs two shifts each day for each 5-day work week. Any jobs not completed at the end of the second shift are carried over to first shift of the next working day. Assume that the contract is going to last for 1 year (52 weeks). Build a simulation model that can assist the company in assessing the risks associated with the new contract.

- 8.31 Parts arrive at a four-workstation system according to an exponential interarrival distribution with a mean of 10 minutes. The workstation A has two machines. The three workstations (B, C, D) each have a single machine. There are three part types, each with an equal probability of arriving. The process plan for the part types are given below. The entries are for exponential distributions with the mean processing time (MPT) parameter given.

Workstation, MPT	Workstation, MPT	Workstation, MPT
A, 9.5	C, 14.1	B, 15
A, 13.5	B, 15	C, 8.5
A, 12.6	B, 11.4	D, 9.0

Assume that the transfer time between arrival and the first station, between all stations, and between the last station and the system exit is 3 minutes. Using the ROUTE, SEQUENCE, and STATION modules, simulate the system for 30,000 minutes and discuss the potential bottlenecks in the system.

- 8.32 Parts arrive to a machine shop according to an exponential distribution with a mean of 10 minutes. Before the parts go into the main machining area, they must be prepped. The preparation area has two preparation machines that are tended by two operators. Upon arrival, parts are assigned to either of the two preparation machines with equal probability. Except for processing times, the two preparation machines operate in the same manner. When a part enters a preparation machine's area, it requires the attention of an operator to set up the part on the machine. After the machine is set up, the machine can process without the operator. Upon completion of the processing, the operator is required to remove the part from the machine. The same operator does all the set ups and part removals. The operator attends to the parts in a FCFS manner. The times for the preparation machines are given in the table below according to a triangular distribution with the provided parameters:

	Preparation Machine	Setup Time	Process Time	Removal Time
1		(8,11,16)	(15,20,23)	(7,9,12)
2		(6,8,14)	(11,15,20)	(4,6,8)

After preparation, the parts must visit the machine shop. There are four machines in the machine shop. The parts follow a specific sequence of machines within the shop. This is determined after they have been prepped. The percentage for each sequence is given in the table below. The #,(min, mode, max) provides the machine number, #, and the parameters for the processing times for a triangular distribution in minutes.

Sequence	%	#,(min, mode, max)	#,(min, mode, max)	#,(min, mode, max)	#,(min, mode, max)
1	12	1,(10.5,11.9,13.2)	2,(7.1,8.5,9.8)	3,(6.7,8,10)	4,(1.8,9,10.3)
2	14	1,(7.3,8.6,10.1)	3,(5.4,7.2, 11.3)	2,(9.6, 11.4, 15.3)	
3	31	2,(8.7,9.9,12)	4,(8.6,10.3,12.8)	1,(10.3, 12.4, 14.8)	3,(8.4,9.7,11)
4	24	3,(7.9,9.3, 10.9)	4,(7.6,8.9,10.3)	3,(6.5,8.3,9.7)	2,(6.7,7.8,9.4)
5	19	2,(5.6,7.1,8.8)	1,(8.1, 9.4, 11.7)	4,(9.1, 10.7, 12.8)	

The transfer time between the preparation area and the first machine in the sequence, between all machines, and between the last machine and the system exit follows a triangular distribution with parameters 2, 4, 6 minutes.

- (a) Run the model for 200,000 minutes. Report average and 95% half-width statistics on the utilization of the preparation operator, the utilization of the preparation machines, the utilization of the job shop machines 1–4, number of parts in the system, and time spent in the system (in minutes).
  - (b) Recommend a warm-up period for the total time spent in the system. Show your work to justify your recommendation.
  - (c) Where is the bottleneck for this system? What would you recommend to improve the performance of this system?
- 8.33 A particular stock keeping unit (SKU) has demand that averages 14 units per year and is Poisson distributed. That is, the time between demands is exponentially distributed with a mean of 1/14 years. Assume that 1 year = 360 days. The inventory is managed according to a  $(r, Q)$  inventory control policy with  $r = 3$  and  $Q = 4$ . The SKU costs \$150. An inventory carrying charge of 0.20 is used and the annual holding cost for each unit has been set at  $0.2 * \$150 = \$30$  per unit per year. The SKU is purchased from an outside supplier and it is estimated that the cost of time and materials required to place a purchase order is about \$15. It takes 45 days to receive a replenishment order. The cost of backordering is very difficult to estimate, but a guess has been made that the annualized cost of a backorder is about \$25 per unit per year.
- (a) Using the analytical results for the  $(r, Q)$  inventory model, compute the total cost of the current policy.
  - (b) Using Arena™, simulate the performance of this system using  $Q = 4$  and  $r = 3$ . Report the average inventory on hand, the cost for operating the policy, the average number of backorders, and the probability of a stock out for your model.

- (c) Now suppose that the lead-time is stochastic and governed by a lognormal distribution with a mean of 45 days and a standard deviation of 7 days. What assumptions do you have to make to simulate this situation? Simulate this situation and compare/contrast the results with parts (a) and (b).
- 8.34 Simulate Exercise 8.34.b using Arena<sup>TM</sup> and OptQuest to recommend optimal values for the  $(r, Q)$ . Use a range of 1 to 6 for the reorder point and a range of 1 to 12 for the reorder quantity.
- 8.35 Suppose the amount demanded for each customer is governed by a geometric random variable with parameter  $p = 0.9$ . What assumptions do you have to make to simulate this situation? How will you handle the possibility of a customer order taking the inventory position significantly below the reorder point? What if ordering  $Q$  does not get the inventory position above the reorder point? How will you process a demand for two items when you only have one item on the shelf?
- 8.36 Reconsider Exercise (8.34). Suppose backordering is not allowed. That is, customers who arrive when there is not enough to fill their demand are lost. Suppose the amount demanded for each customer is governed by a geometric random variable with parameter  $p = 0.9$ . Develop an Arena<sup>TM</sup> model for this situation. Estimate the expected number of lost sales per year for this situation.
- 8.37 Simulate Exercise (8.37) using Arena<sup>TM</sup> and OptQuest to recommend optimal values for the  $(r, Q)$  when the value of a lost sale is set at \$40.
- 8.38 In a continuous review  $(s, S)$  inventory control system, the system operates essentially the same as an  $(r, Q)$  control policy except that when the reorder point,  $s$ , is reached, the system instead orders enough inventory to reach a maximum inventory level of  $S$ . Thus, the amount of the order will vary with each order. Repeat the analysis of Exercise 8.34.b assuming that the SKU is controlled with a  $(s, S)$  policy. Assume that  $s = 4$  and  $S = 7$ . Develop and simulate this system and compare the cost results to the results of Exercise 8.34.b.
- 8.39 Suppose in Exercise (8.39) that the amount demanded for each customer is governed by a geometric random variable with parameter  $p = 0.9$ . In addition, assume that the backorder queue is filled on a FCFS basis. Develop a model and simulate this system.
- 8.40 In a periodic review  $(R, S)$  inventory control system, the time between reviews of the inventory position is given by the parameter  $R$ . For example, if  $R = 1$  week, at the end of each week, the inventory position is reviewed. If the inventory position is less than  $S$ , the amount necessary to bring the inventory position back up to  $S$  is ordered. Repeat the analysis of Exercise (8.32) assuming the SKU is controlled with an  $(R, S)$  policy. Use  $R = 1/2$  year and  $S = 7$ . Develop and simulate this system and compare the cost results to the results of Exercise 8.34.b.
- 8.41 This exercise considers the periodic review  $(R, s, S)$  inventory control system with back ordering. This control policy is a combination of the  $(s, S)$  and  $(R, S)$  policies. See Exercises (8.39) and (8.41). The time between reviews of the inventory position is given by the parameter  $R$ . When the inventory position is reviewed and the inventory position is less than  $s$ , then the amount necessary to bring the inventory position back up to  $S$  is ordered. Repeat the analysis of Exercise 8.34.b assuming the SKU is controlled with a  $(R, s, S)$  policy. Use  $R = 1/2$  year,  $s = 3$ , and  $S = 7$ . Develop and simulate this system and compare the cost results to the results of Exercise 8.34.b.

- 8.42 Consider Exercise (8.34). Suppose that the replenishment orders are served by a production facility that acts like a single-server queuing system. The production time is lognormally distributed with a mean of 30 days and a standard deviation of 7 days. The transport delay is a constant of 1 day. Simulate this situation and compare/contrast the results with Exercise 8.34.b. What happens if the demand for the item doubles?
- 8.43 Consider a shop that produces and stocks items. The items are produced in lots. The demand process for each item is Poisson with the rate given below. Backordering is permitted. The pertinent data for each of the items is given in the following table. The carrying charge on each item is 0.20. The lead-time is considered constant with the value provided in the table. Assume that there are 360 days in each year.

Item	1	2	3	4	5	6
Demand Rate (units per year)	1000	500	2000	100	50	250
Unit cost (dollars per unit)	20	100	50	500	1000	200
Lead-time mean (days)	2	5	5	6	14	8

Suppose that management desires to set the optimal  $(r, Q)$  policy variables for each of the items so as to minimize the total holding cost and ordering costs. Assume that it costs approximately \$5 to prepare and process each order. In addition, they desire to have the system fill rate to be no lower than 80% and no individual item's fill rate to be below 60%. They also desire to have the average customer wait time (for any part) to be less than or equal to 3 days. Simulate this system and use OptQuest to find an optimal policy recommendation for the items. You might start by analyzing the parts individually to have initial starting values for the optimization search.

- 8.44 Consider the model for the multi-echelon inventory system presented within the chapter. Suppose that the retail locations do not allow backordering. That is, customers who arrive when there is not enough to fill their demand are lost. In addition, suppose that the amount demanded for each customer is governed by a geometric random variable with parameter  $p = 0.9$ . The warehouse and the retailers control the SKU with  $(s, S)$  policies as shown in the following table.

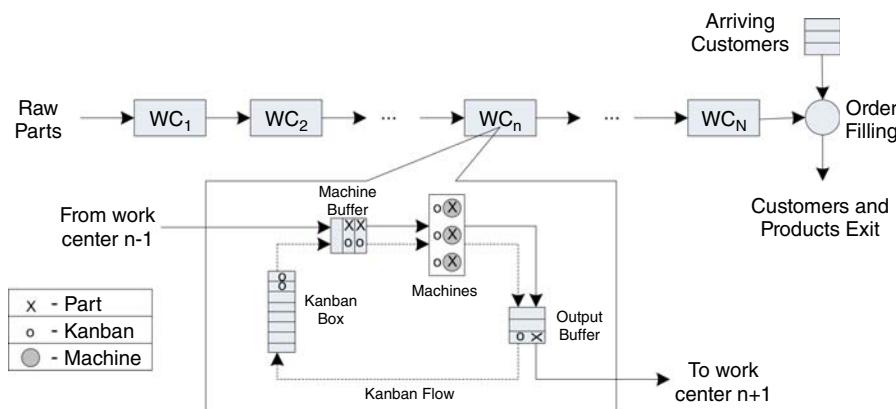
	Policy $(s, S)$	Demand Poisson ( $\lambda$ )
Retailer 1	(1,3)	(0,2)
Retailer 2	(1,5)	(0,1)
Retailer 3	(2,6)	(0,3)
Retailer 4	(1,4)	(0,4)
Warehouse	(4,16)	

The overall rate to the retailers is 1 demand per day, but individually they experience the rates as shown in the table. In other words, the retailers are no longer identical in terms of the policy or the demand that they experience. The lead-times remain as 1 day for the warehouse and for each retailer. The warehouse allows backordering. Develop an Arena™ model for this situation. Estimate the expected number of lost sales per year for each retailer, the average amount of inventory on hand at the retailers

and at the warehouse, the fill rate for each retailer and for the warehouse, and the average number of items backordered at the warehouse. Run the model for 10 replications of 3960 days with a warm-up period of 360 days.

- 8.45 Consider the multi-echelon inventory system presented within the chapter. Suppose multiple types of items can be located at each retailer and at the warehouse. Describe the basic model changes that would need to be made to address this situation.
- 8.46 Consider the multi-echelon inventory system presented within the chapter. Suppose that the system is being expanded so that there are multiple item types (as in Exercise (8.44) and there are now two warehouse locations. Each warehouse supplies two of the retailers. In addition, each warehouse is supplied by a national distribution center. Describe the basic model changes that would need to be made to address this situation.
- 8.47 This problem examines a single item, multistage, serial production system using a kanban control system. The word kanban is Japanese for card. A kanban is simply a card that authorizes production. Kanban-based control systems are designed to limit the amount of inventory in production. While there are many variations of kanban control, this problem considers a simplified system with the following characteristics.
- A series of work centers.
  - Each work center consists of one or more identical machines with an input queue feeding the machines that is essentially infinite.
  - Each work center also has an output queue where completed parts are held to await being sent to the next work center.
  - Each work center has a schedule board where requests for new parts are posted.
  - Customer orders arrive randomly at the last work center to request parts. If there are no parts available, then the customers will wait in a queue until a finished part is placed in the output buffer of the last work center.

The following figure illustrates a production flow line consisting of kanban-based work centers.



Raw parts enter at the first work center and proceed, as needed, through each work center until eventually being completed as finished parts at the last work center. Finished parts are used to satisfy end customer demand.

A detailed view of an individual kanban-based work center is also shown in the figure. Here parts from the previous work center are matched with an available kanban

to proceed through the machining center. Parts completed by the machining center are placed in the output buffer (along with their kanban) to await demand from the next work center.

An individual work center works as follows:

- Each work center has a finite number of kanbans,  $k_n \geq 1$ .
- Every part must acquire one of these kanbans in order to enter a given work center and begin processing at the machining center. The part holds the kanban throughout it stay in the work center (including the time it is in the output buffer).
- Unattached kanbans are stored on the schedule board and are considered as requests for additional parts from the previous work center.
- When a part completes processing at the machines, the system checks to see if there is a free kanban at the next work center. If there is a free kanban at the next work center, the part releases its current kanban and seizes the available kanban from the next work center. The part then proceeds to the next work center to begin processing at the machines. If a kanban is not available at the next work center, the part is place in the output buffer of the work center to await a signal that there is a free kanban available at the next station.
- Whenever a kanban is released at the work center, it signals any waiting parts in the output buffer of its feeder work center. If there is a waiting part in the output buffer when the signal occurs, the part leaves the output buffer, releasing its current kanban, and seizing the kanban of the next station. It then proceeds to the next work center to be processed by the machines.

For the first work center in the line, whenever it needs a part (i.e., when one of its kanbans is released) a new part is created and immediately enters the work center. For the last work center, a kanban in its output buffer is not released until a customer arrives and takes away the finished part to which the kanban was attached. Consider a three work center system. Let  $k_N$  be the number of kanbans and let  $c_n$  be the number of machines at work center  $n$ ,  $n = 1, 2, 3$ . Let us assume that each machine's service time distribution is an exponential distribution with service rate  $\mu_n$ . Demands from customers arrive to the system according to a Poisson process with rate  $\lambda$ . As an added twist assume that the queue of arriving customers can be limited. In other words, the customer queue has a finite waiting capacity, say  $C_q$ . If a customer arrives when there are  $C_q$  customers already in the customer queue, they do not enter the system. Build a simulation model to simulate the following case:

$c_1$	$c_2$	$c_3$
1	1	1
$k_1$	$k_2$	$k_3$
2	2	2
$\mu_1$	$\mu_2$	$\mu_3$
3	3	3

where  $\lambda = 2$  and  $C_q = \infty$  with all rates per hour. Estimate the average hourly throughput for the production line. That is, the number of items produced per hour. Run the simulation for 10 replications of length 15,000 hours with a warm-up period of 5000 hours. Suppose  $C_q = 5$ , what is the effect on the system.

---

# 9

---

## ENTITY MOVEMENT AND MATERIAL-HANDLING CONSTRUCTS

### LEARNING OBJECTIVES

- To be able to model constrained entity transfer with resources.
- To be able to model constrained entity transfer with transporters.
- To be able to model systems involving conveyors.
- To be able to model systems involving automatic-guided vehicles.
- To be able to perform basic animation for entity transfer situations.

### 9.1 INTRODUCTION

Chapter 8 introduced the concept of unconnected entity movement through the use of the ROUTE and STATION modules. The ROUTE module models entity movement between stations as a simple time delay. The entities can be imagined as “having little feet” that allow them to move from one station to another. That is, the entity is able to move itself. If the entity is a person (e.g., a patient in a clinic), this representation makes sense; however, if the entity is a part in a manufacturing system, this representation begins to breakdown. For example, in the test and repair situation of Chapter 8, the parts moved between stations with a time delay. But how did they physically move? One way to think about this is that there were workers always available to move the parts between the stations. If there is *always* a worker available, then it is as if there is an infinite supply of workers. Thus, whenever a part must be moved from one station to another, the part uses one of the workers to make the movement. Since there is an infinite supply of workers, this is the same as the part moving

itself (i.e., having little feet) and only the time delay for moving between the stations is relevant.

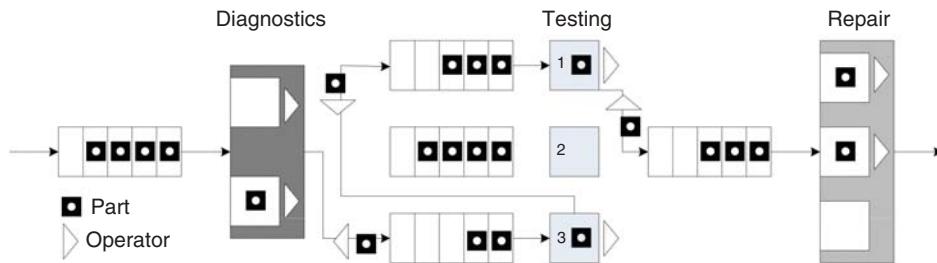
In many situations, modeling transfers with a delay is perfectly reasonable, especially if you are not interested in how the entities moved (only that they moved). However, in many situations, the movement of entities can become constrained by the lack of availability of the transfer mechanism. For example, the movement of parts may require that the parts be placed on a pallet and that a fork lift be used to move the parts. At any point in time, there may not be enough fork lifts available and thus the parts may have to wait for a fork lift to become available. When the potential for waiting for transport is significant, you might want to model with more detail the “how” behind the entity transfer. In addition, since movement can be a significant part of an operation, the design of the material movement system may be the main focus of the modeling effort.

This chapter explores the various constructs available within Arena™ to facilitate the modeling of the physical movement of entities between stations. The chapter begins by describing how to model transfers using resources. In this case, the transfer delay is accompanied by the use of a resource. Then, Section 9 presents how Arena™ facilitates resource-constrained movement using the TRANSPORTER module and its accompanying constructs. A transporter is a special kind of resource in Arena™ that can move. Since not all movement is as freely moving through space as people walking or fork trucks moving, Arena™ provides constructs for modeling entity movement when the space between the locations becomes an important aspect of the modeling. Section 9 indicates how conveyors can be represented within Arena™ and how they represent the space between stations. Then, in Section 9, the modeling of transporters will be revisited to understand how to model the situation where the transporters may compete for space while moving. This will involve the modeling of the space between stations as a fixed path (i.e., like a road network with intersections) As usual, these concepts will be illustrated with example models.

## 9.2 RESOURCE-CONSTRAINED TRANSFER

When an entity requires to use something to complete the movement between stations, Arena’s RESOURCE module can be used to model the situation. In this situation, the physical (e.g., distance) aspects of the movement are not of interest, only the time that may be constrained by the availability of the transport mechanism. To illustrate this modeling, let us revisit the test and repair shop from Chapter 8.

Recall that in the test and repair shop, parts follow one of four different test plans through the shop. Each part first goes to the diagnostic station where it determines the sequence of stations that it will visit via an assignment of a test plan. After being diagnosed, it then proceeds to the first station in its test plan. In the example of Chapter 8, it was assumed that a worker (from somewhere) was always available to move the entity to the next station and that the transfer time took between 2 and 4 minutes uniformly distributed. The diagnostic station had two diagnostic machines and each test station had one testing machine. Finally, the repair station had three workers who performed the necessary repairs on the parts after testing. An *implicit* assumption in the model was that there was a worker staffing each of the two diagnostic machines (one worker for each machine) and that there was a worker assigned to each test station. The modeling of the workers was not a key component of the modeling so that such an implicit assumption was fine.



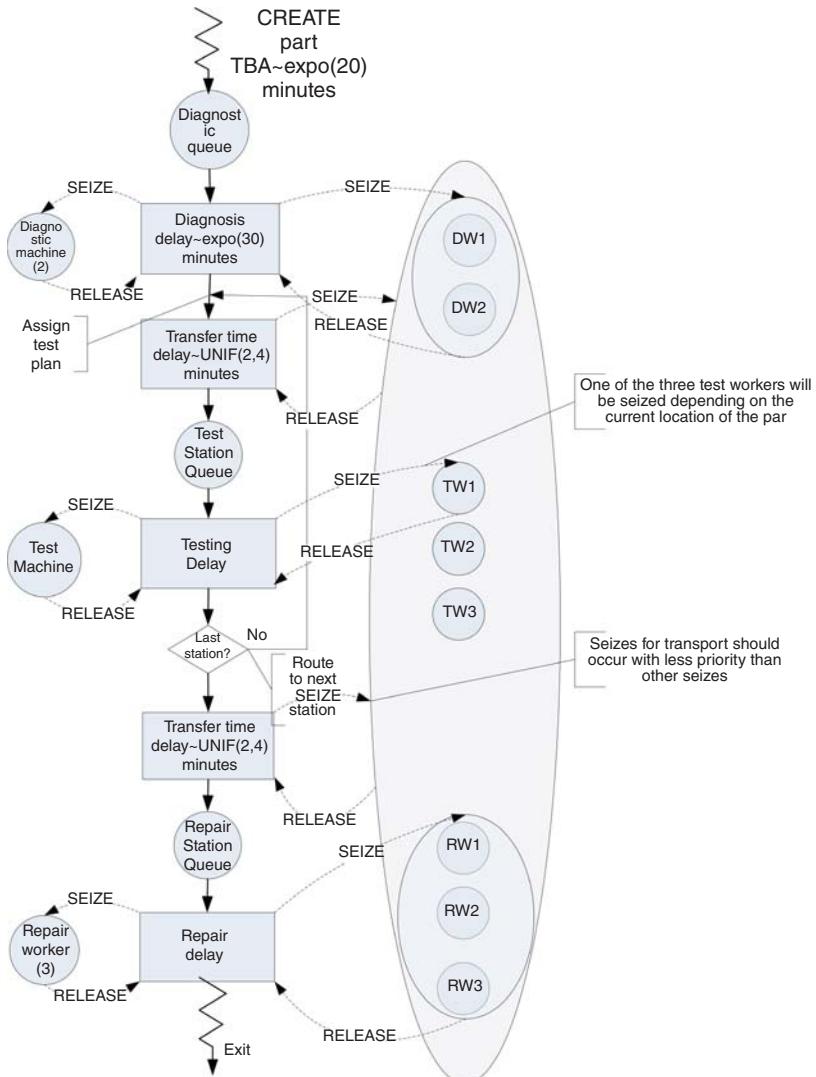
**Figure 9.1** Test and repair shop with workers providing the movement.

In this section, the use of the workers to move the entities and to staff the stations will be explicitly modeled. Assume that there are two workers at the diagnostic station, one worker per testing station, and three workers at the repair station. Thus, there are a total of eight workers in the system. For simplicity, assume that any of these eight workers are capable of moving parts between the stations. For example, when a part completes its operation at the diagnostic station, any worker in the system can carry the part to the next station. In reality, it may be useful to assign certain workers to certain transfers (e.g., diagnostic workers move parts to the part's first station); however, for simplicity, these issues will be ignored and any worker will be allowed to do any transport in this example. This also requires that any worker is capable of noticing that a part needs movement. For example, perhaps the part is put in a basket and a light goes on indicating that the part needs movement. When a part requires movement, it will wait for the next available idle worker to complete the movement. In this situation, a worker may be busy tending to a part in process at a station or the worker may be busy moving a part between stations. Figure 9.1 illustrates the new situation for the test and repair shop involving the use of workers.

Since workers are required for the processing of parts at the stations and they might have to perform the movement of parts between stations, the workers must be shared between two activities. Thus, when a worker completes one activity a mechanism is needed to indicate which activity should proceed next. A simple mechanism is to assign a priority to one of the tasks. Thus, it seems reasonable to assume that parts waiting for processing at a station are given priority over parts that require movement between stations.

Figure 9.2 illustrates an activity diagram for the situation where the workers are called for the transport. In the figure, each worker is indicated individually. For example, DW1 refers to worker 1 at the diagnostic station. In the figure, the visitation of each part to a different test station is illustrated with a loop back to the transfer time after the testing delay. Thus, the figure represents all three test stations with the test station queue, testing delay, and test machine combination. Unfortunately, this does not explicitly indicate that a test worker is assigned to each test station individually. In particular, the other resources marked TW2 and TW3 should technically have seize and release arrows associated with them for the testing activity at a particular station.

It should be clear from the figure that three sets of resources will be required in this model. A resource set should be defined for the diagnostic workers with two members. A resource set should be defined for the three repair workers. Finally, a resource set should be defined to hold each of the workers DW1, DW2, TW1, TW2, TW3, RW1, RW2, RW3 who are available for transporting parts. When a part requires diagnostics, it will seize one of the workers in the diagnostic workers set. When a part requires testing, it will seize the appropriate test worker for its current station. Finally, at the repair station, the part will seize



**Figure 9.2** Activity diagram for revised test and repair situation.

one of the repair workers. In order to receive transport, the part will seize from the entire worker set.

### 9.2.1 Implementing Resource-Constrained Transfer

Based on Figure 9.2, the basic pseudo-code for resource-constrained transfer should be something like that shown in Exhibit 9.1. This logic assumes that each worker has been placed in the appropriate sets. At the diagnostic station, both the machine and a diagnostic worker are required. When the part completes the processing at the diagnostic station, the part seizes a worker from the overall set of workers and routes with a transfer delay to the appropriate testing station.

**Exhibit 9.1** Pseudo-Code for Resource Constrained Transfer

```

CREATE part
SEIZE 1 diagnostic machine
SEIZE 1 diagnostic worker from diagnostic worker set
DELAY for diagnostic time
RELEASE diagnostic machine
RELEASE diagnostic worker
ASSIGN test plan sequence
SEIZE 1 worker from worker set
ROUTE for transfer time by sequence to STATION Test

STATION Test
RELEASE worker
SEIZE appropriate test machine
SEIZE appropriate test worker
DELAY for testing time
RELEASE test machine
RELEASE test worker
SEIZE 1 worker from worker set
DECIDE
    IF not at last station THEN
        ROUTE for transfer time by sequence to STATION Test
    ELSE
        ROUTE for transfer time by sequence to STATION Repair
    END IF
END DECIDE

STATION Repair
RELEASE worker
SEIZE repair worker from repair worker set
DELAY for repair time
RELEASE repair worker
RECORD statistics
DISPOSE

```

After arriving at the test station, the part releases the worker who performed the transport and proceeds with the seizing of the test machine and worker. Again, prior to the routing to the next station, a worker from the overall set of workers is seized. Notice also that the worker is released after the part is transferred to the repair station.

The combination of logic (SEIZE, ROUTE, STATION, RELEASE) is very common in entity transfer modeling. While this logic can be directly implemented with the corresponding Arena™ modules, Arena™ has provided two additional modules that enable a wide range of transfer modeling options through dialog box entries. These modules are the ENTER and LEAVE modules of the Advanced Transfer panel. The ENTER module represents a number of different modules based on how its dialog box entries are completed. Essentially, an ENTER module represents the concepts of a STATION, a delay for unloading, and the releasing of the transfer method. Within the LEAVE module, the user can specify a delay for loading, how the entity will be transferred out (e.g., resource, conveyor, and transporter), and the routing of the entity. Now let us take a look at how to modify the test and repair model from Chapter 8 to use these new constructs.

Starting with a copy of the *RepairShop.doe* file from Chapter 8, the first step is to define 8 individual resources for the diagnostic workers (2), the test workers (1 for each station), and the repair workers (3). This is illustrated in Figure 9.3, where the workers have been added

Resource - Basic Process			
	Name	Type	Capacity
1	DiagnosticMachine	Fixed Capacity	2
2	TestMachine1	Fixed Capacity	1
3	TestMachine2	Fixed Capacity	1
4	TestMachine3	Fixed Capacity	1
5	RepairWorker1	Fixed Capacity	1
6	RepairWorker2	Fixed Capacity	1
7	RepairWorker3	Fixed Capacity	1
8	DiagnosticWorker1	Fixed Capacity	1
9	DiagnosticWorker2	Fixed Capacity	1
10	TestWorker1	Fixed Capacity	1
11	TestWorker2	Fixed Capacity	1
12	TestWorker3	Fixed Capacity	1

Double-click here to add a new row.

Figure 9.3 Resources for test and repair shop.

Set - Basic Process			
	Name	Type	Members
1	DiagnosticWorkers	Resource	2 rows
2	RepairWorkers	Resource	3 rows
3	Workers	Resource	8 rows

Double-click here to add a new row

	Resource Name
1	TestWorker1
2	TestWorker2
3	TestWorker3
4	DiagnosticWorker1
5	DiagnosticWorker2
6	RepairWorker1
7	RepairWorker2
8	RepairWorker3

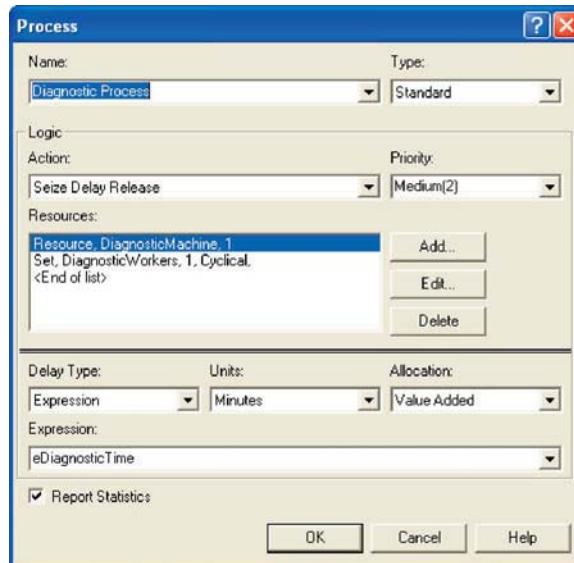
Double-click here to add a new row

Figure 9.4 Resource sets for test and repair shop.

to the RESOURCE module. Notice that the repair workers have been changed from having a single resource with capacity 3 to three resources each with capacity 1. This will enable all eight workers represented as resources to be placed in an overall worker(resource)set.

Use the SET module on the Basic Process panel to define each of the three sets required for this problem. Figure 9.4 illustrates the Workers set within the SET module. The *DiagnosticWorkers* and *RepairWorkers* sets are defined in a similar manner using the appropriate resources. Since there was no preference given in the use of the resources within the sets, you can just list them as shown in the figure. Recall that for the preferred resource selection rule, the order of the resources matters. The cyclical rule is used in this model.

Now that the new resources and their sets have been defined, you need to update each of the PROCESS modules in order to ensure that the appropriate resources are seized and



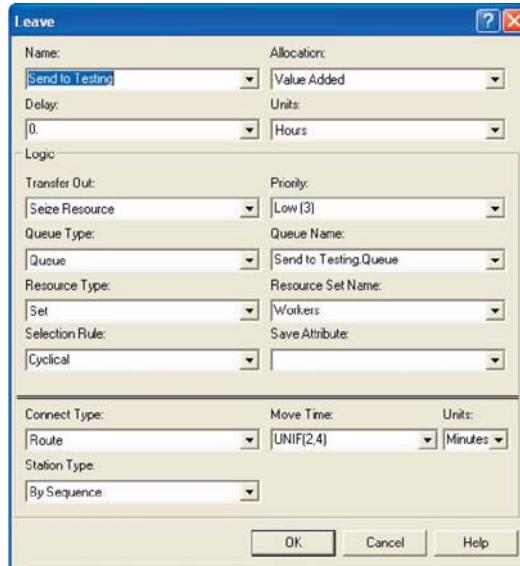
**Figure 9.5** Seizing within the diagnostic process.

released. This is illustrated in Figure 9.5. In the Diagnostic Process dialog, the SEIZE, DELAY, RELEASE option has been used with the *list* of resources required. In this case, the part needs both 1 unit of the diagnostic machine and 1 unit from the *DiagnosticWorkers* set. Each of the testing stations is done in a similar manner by first seizing the test machine and then seizing the test worker. The repair station is done in a similar manner, except there is only the seizing of 1 unit from the *RepairWorkers* set. In all cases, the cyclical rule is used with the level of medium for the seize priority. If you are following along with the model building process, you should update each of the PROCESS modules as described. The completed model is found in the file *RepairShopResourceConstrained.doe* that accompanies this chapter.

Now the ENTER and LEAVE modules can be specified. The STATION modules associated with the three testing stations and the repair station will be replaced with appropriately configured ENTER modules. In addition, the ROUTE modules will be replaced with LEAVE modules for routing to the testing stations, between the testing stations, and to the repair station. Let us start with specifying how to leave the diagnostic station. Delete the ROUTE module associated with the diagnostic station and drag a LEAVE module into its place. Then fill out the LEAVE module as shown in Figure 9.6.

When you first open the LEAVE module, it will not look as shown in the figure. As can be seen in the figure, it is divided into three components: Delay, Logic (Transfer Out), and Connect Type. The Delay text field allows a time delay to be specified after getting the transfer out option. This can be used to represent a loading time. In this case, the loading time will be ignored (it takes negligible time for the worker to pick up the part). The transfer out logic allows the specification of the use of a transporter, a conveyor, a resource, or no constrained transfer (none). In this case, the seize resource option is required.

After selecting one of the resource-constrained transfer options, additional options will become available. Because the transfer is constrained, the entity must have a place to wait for the transfer if the resource is not immediately available. Thus, there is the option to



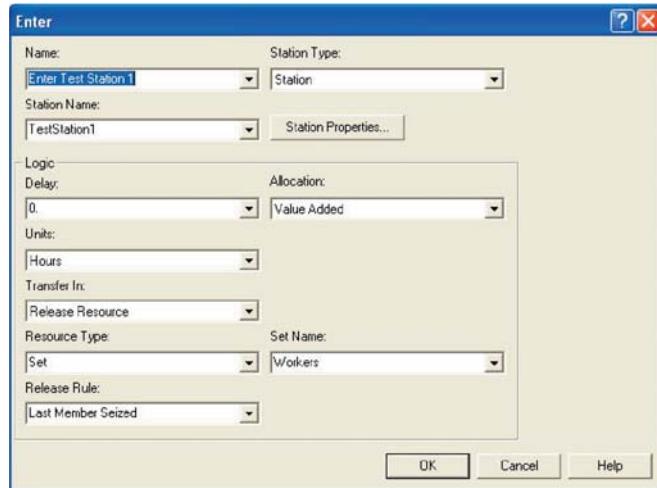
**Figure 9.6** LEAVE module for test and repair shop example.

specify a queue for the entity. You can think of this queue as the output bin for the station. Just like in the PROCESS module, you can directly seize a resource or seize a resource from a resource set. In this case, one worker will be seized from the Workers set. Notice that a low seize priority has been specified. Since this priority is lower than that used by the parts when seizing with the PROCESS modules, the PROCESS modules will have higher priority when parts are waiting for processing over those waiting for transfer.

Finally, the Connect Type option within the LEAVE module is just like the corresponding options in the ROUTE module. In this case, the entity can be routed by sequence with a routing delay of UNIF(2,4) minutes. Each of the other ROUTE modules in the previous model should be replaced with LEAVE modules like that shown in Figure 9.6. To do this replacement quicker, you can simply delete the ROUTE modules and copy/paste LEAVE modules as per Figure 9.6. You should make sure to change the name of the modules after the cut/paste operation, since no two Arena™ modules can have the same name.

Now, the ENTER module can be specified. The ENTER module for test station 1 is given in Figure 9.7. Again, the ENTER module allows a composition of a number of other modules by having a Station Name and Logic options. Within the Logic options, you can specify a delay and the transfer in options. The Delay option represents a delay that occurs prior to the transfer in option. This is commonly used to represent a delay for unloading the transferring device. In this model, the time to unload (put down the part) is assumed to be negligible for the worker.

The transfer in option indicates how the entity should react with respect to the transfer mechanism. There are four options (Free Transporter, Exit Conveyor, Release Resource, and None). A simple unconstrained transfer uses the *None* option. In this model, the release resource option should be used. Since the worker who was seized for the transfer was part of a set, you can select the set option for the resource type. The release rule specifies how the resource will be released. In this case, the last member seized option should be used.



**Figure 9.7** ENTER module for test and repair shop example.

User Specified		
Tally		
Expression	Average	Half Width
ProbLTContractLimit	0.7204	0.04
SystemTimeStat	408.51	25.74

**Figure 9.8** Probability of meeting the contract requirements.

This means that the most recent member of the Workers set that the part seized will be released.

Similar to what was done in Chapter 8, the model was run for 10 replications of 4160 hours to examine the effect on the performance measures. As can be seen in Figure 9.8, the probability of meeting the contract specification has been reduced by about 10% (from 82.51% in the example of Chapter 8 to 72.04% in this example). This is due to the increase in the average system time.

As can be seen in Figure 9.9, the time spent waiting in the station's output buffer for pick up ranges from 13 to 15 minutes. Since there are four queues, this time has added significantly to the system time of the parts. In Figure 9.10, the utilization of the resources is relatively high (near 90% in most cases).

Because of the increased risk of not meeting the contract specifications for the system modeled with the more realistic use of workers to transfer parts, it might be useful to consider alternative ways to transfer the parts between stations. The next section examines the use of dedicated workers modeled as transporters to perform this work. Before proceeding with that modeling, the animation needs to be enhanced.

<b>Time</b>		
Waiting Time	Average	Half Width
Diagnostic Process.Queue	41.9538	2.54
Leave Test Station 1.Queue	13.0437	2.88
Leave Test Station 2.Queue	14.0260	3.04
Leave Test Station 3.Queue	13.3562	2.98
Repair Process.Queue	25.3725	3.23
Send to Testing.Queue	14.9525	3.16
Testing Process 1.Queue	53.0058	3.49
Testing Process 2.Queue	40.5793	2.73
Testing Process 3.Queue	53.7482	6.07
<b>Other</b>		
Number Waiting	Average	Half Width
Diagnostic Process.Queue	2.0944	0.14
Leave Test Station 1.Queue	0.7325	0.17
Leave Test Station 2.Queue	0.5285	0.12
Leave Test Station 3.Queue	0.6676	0.15
Repair Process.Queue	1.2663	0.17
Send to Testing.Queue	0.7478	0.16
Testing Process 1.Queue	2.9724	0.21
Testing Process 2.Queue	1.5285	0.11
Testing Process 3.Queue	2.6842	0.32

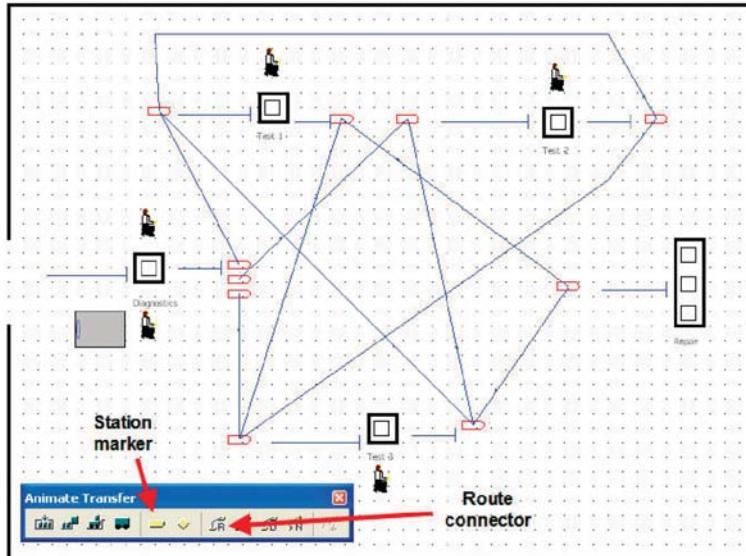
**Figure 9.9** Queue statistics for test and repair with resource-constrained transfer.

Instantaneous Utilization	Average	Half Width
DiagnosticMachine	0.7484	0.01
DiagnosticWorker1	0.8399	0.01
DiagnosticWorker2	0.8399	0.01
RepairWorker1	0.9223	0.01
RepairWorker2	0.9228	0.01
RepairWorker3	0.9224	0.01
TestMachine1	0.8545	0.01
TestMachine2	0.7773	0.01
TestMachine3	0.8607	0.01
TestWorker1	0.9225	0.01
TestWorker2	0.8735	0.01
TestWorker3	0.9229	0.01

**Figure 9.10** Utilization for test and repair with resource-constrained transfer.

### 9.2.2 Animating Resource-Constrained Transfer

If you ran the model of the previous section with the animation turned on, you would notice the queues for the PROCESS modules and the ROUTE modules. Unfortunately, you cannot see the entities during the transfer between stations. In this section, you will augment the basic flowchart animation with route animation, so that the flow of the entities between the stations will be visible. The animation of the stations will also be updated. Unfortunately, to really represent the use of the resources well within the animation, more effort would need to be expended on the animation than is useful at this time. For example, workers can be busy tending a process or walking. To properly represent this, the definition of additional



**Figure 9.11** Final animation for resource-constrained transfer.

resource states and their animation would need to be discussed. As will be covered in the next section, the use of transporters will greatly assist in that regard. Thus, for simplicity, the movement of the entities between the stations will have a rather crude representation of the resource usage.

Figure 9.11 shows the final animation for the model. The completed model is available in the file *RepairShopResourceConstrainedTransferWithAnimation.doe*. The key new animation concept that needs to be introduced is the Animate Transfer toolbar. Animate Transfer toolbar. Notice that in the figure, the toolbar has been detached. The two key buttons that will be used are the Station Marker button and the Route connector button.

An overview of how to create this animation starting from the file *RepairShopResource-ConstrainedTransfer.doe* is as follows:

1. Use Arena's drawing toolbar to place the dark lines for the outline of the repair shop and for the dark rectangles to represent the stations. Use the text label button to label each of the station areas.
2. Use the Resource button on the Basic Animate toolbar to place each of the resources (Diagnostic machines, test machines, repair workers, diagnostic workers, and test workers). In the figure, the standard animation for resources was used to represent the diagnostic machines, the test machines, and the repair workers. The picture library (as described in Chapter 4) was used to assign a seated worker to the idle picture and a worker who looks to be walking to the busy state of the diagnostic and test workers.
3. Delete the queues from the flowchart area and readd them using the Queue button of the Basic Animate toolbar in the locations indicated in the figure. This process is similar to what was described in Chapter 4.
4. Use the Variable button on the Basic Animate toolbar to place the variable animation to show the number of diagnostic machines that are currently busy.

**TABLE 9.1 Distribution of Test Plans**

Test Plan	Percentage of Parts, %	Sequence
1	25	2,3,2,1
2	12.5	3,1
3	37	1,3,1
4	25	2,3

**TABLE 9.2 From/To Indicator Table**

Station	Destination			
	Test 1	Test 2	Test 3	Repair
Origin	Diagnostics	Yes	Yes	Yes
	Test 1	No	No	Yes
	Test 2	Yes	No	No
	Test 3	Yes	Yes	Yes

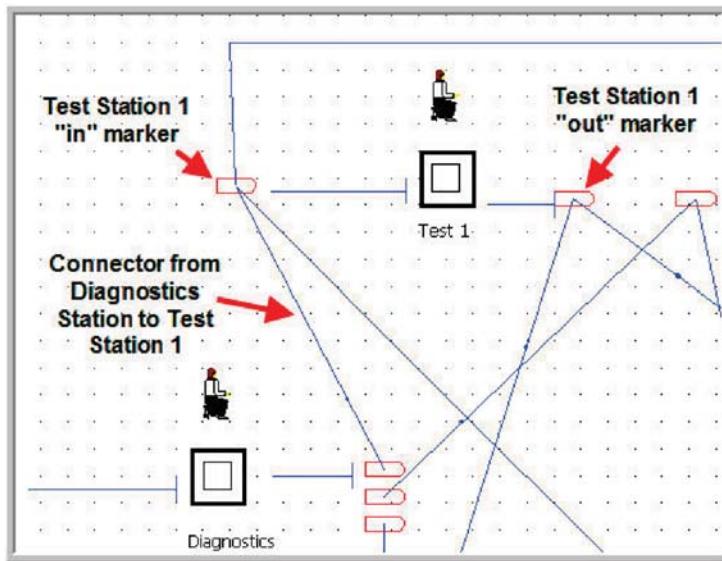
Now, the animation of the routes taken by the parts as they move from station to station can be described. Recall that the test plan sequences are as shown in Table 9.1.

From the test plans, a from/to indicator table can be developed, which shows whether or not a part will go from a given station to another station. In Table 9.2, “Yes” means that the station sends a part from the origin station to the destination station. This table indicates whether or not a route needs to be placed between the indicated stations.

To animate the routes between these stations, you need to place station markers and connect the appropriate stations using the route connectors. To place a station marker, click on the station button of the Animate Transfer toolbar. Your cursor will turn into a cross-hair and you can place the marker in the model window where appropriate. As indicated in Figure 9.12, you can place a station marker for coming into the station and exiting the station. This helps in visualizing the flow through the station. Station markers and connections will have to be placed for all the connections indicated in Table 9.2.

If you miss a connection, you just will not see the part moving between the associated stations. To connect existing station markers, click on the Route button. The cursor will turn into a cross-hair and you can select the appropriate station markers. Notice that the station marker will become highlighted as you connect. If you do not have existing station markers, after clicking the Route button, your first click will lay down a station marker and then you move your cursor to the location for the second station marker. This process takes a little patience and practice. If you are having trouble selecting stations, you should consider using the View toolbar to zoom in to help with the process. In addition, if you have to repeat the same step over and over, you can right-click on the mouse after a step (e.g., placing a station marker) and choose the Repeat Last Action item from the pop-up menu. By using this, you do not have to repeatedly press the toolbar button.

If you run the model with the animation on, you will see the parts moving along the route connectors between the stations, the resource states changing, and the queues increasing and decreasing. If you watch the animation carefully, you should notice that the repair workers become busy when either working on a part or transporting the part. As previously mentioned, it would be nice to separate that out in the animation. It can be done, but



**Figure 9.12** Station markers and route connections.

the STATESET module would need to be used for the resources. You should consult the Arena<sup>TM</sup> Users Guide or help system for more information on the STATESET module.

Having the workers both tend to the processing on the machines and move the parts between the stations has caused the system time to increase. The next section examines the use of dedicated workers to transport the parts between stations using the TRANSPORTER module.

### 9.3 CONSTRAINED TRANSFER WITH TRANSPORTERS

A *transporter* refers to one or more identical transfer devices that can be allocated to an entity for the purpose of transferring entities between stations. Two types of transporters are available within Arena<sup>TM</sup>:

**free path transporter.** The travel time between stations depends on distance and speed.

**guided path transporter.** The travel time between stations depends on distance and speed, and the movement of the transporter is restricted to a predefined network of intersections and links.

The standard delay option of the ROUTE or LEAVE modules assumes no resource-constrained transfer. As indicated in the last section, the situation of a general resource being used for the transfer can be easily modeled. In that situation, the time of the transfer was specified. In the case of transporters, resource-constrained transfer is still being modeled, but in addition, the physical distance of the transfer must be modeled. In essence, rather than specifying a time to transfer, you must specify a velocity and a distance for the transfer. From the velocity and distance associated with the transfer, the time of the transfer can be computed. In these situations, the physical device and its

movement (with or without the entity) through the system are of key interest within the modeling.

To model with transporters, Arena™ provides the following modules on the Advanced Transfer panel:

**ALLOCATE** Attempts to allocate a unit of the transporter to the entity. This is like a SEIZE module when using resources. If the transporter is not immediately available for the request, the entity must wait in a queue. ALLOCATE only gives a unit of the transporter to the entity. It does not move the transporter to the entity's location. It changes the status from idle to busy. This is useful when you want to perform other activities before or during the time the empty transporter is moved to the entity's location.

**MOVE** Causes an allocated transporter to move to a particular location. The controlling entity is not moved during this process.

**REQUEST** Attempts to allocate a transporter and then move the transporter to the location of the requesting entity. REQUEST has the net effect of having an ALLOCATE followed by a MOVE module. textit. Causes the entity to be transported to its destination. To contrast TRANSPORT and MOVE, MOVE is moving unloaded and TRANSPORT is moving loaded with the entity. The entity must have the possession of the transporter unit. Before the transporter can be reallocated to another requesting entity, the transporter must be deallocated using the FREE module.

**FREE** Causes the entity to deallocate the transporter. This is similar to releasing a resource.

**HALT** Causes the transporter to stop moving and become inactive. If the transporter is currently busy at the time when an entity enters the HALT module, the status of the transporter is considered busy and inactive until the entity that controls the transporter frees the unit. If the transporter is idle at the time when an entity halts the transporter, it is set to inactive immediately. This is useful in modeling breakdowns during movement. Once halted, the transporter cannot be allocated until it has been activated.

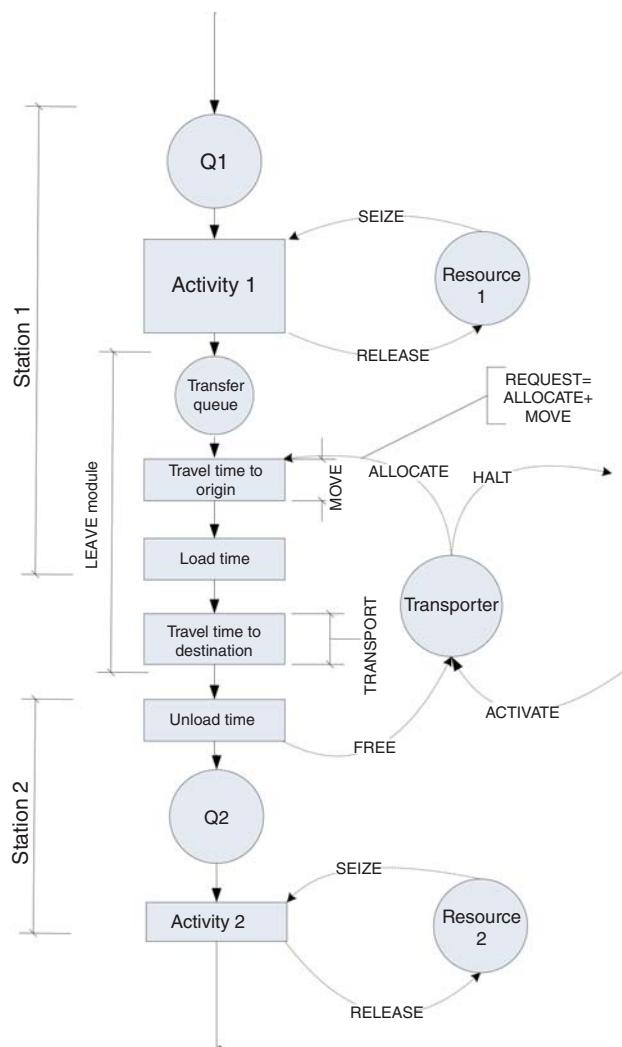
**ACTIVATE** This module causes a halted transporter to become active so that it can then continue its use within the model. It increases the capacity of a previously halted transporter or a transporter that was initially inactive (as defined in the TRANSPORTER module). The transporter unit that is activated will reside at the station location at which it was halted until it is moved or requested by an entity. If an allocation request is pending for the transporter at the time the unit is activated, the requesting entity will gain control of the transporter immediately.

**TRANSPORTER** This data module allows the user to define the characteristics of the transport device, for example, number of transporter units, default velocity, and initial location. In addition, the DISTANCE module associated with the transporter must be specified.

**DISTANCE** This data module allows the user to define the distances (in a consistent unit of measure, e.g., meters) for all the possible moves that the transporter may make. The values for the distances must be nonnegative integers. Thus, if the distance is given in decimals (e.g., 20.2 meters), the distance should either be rounded (e.g., 20 meters) or scaled to integers (202 decameters). In any case, the units of measure for the distances

must be consistent. The DISTANCE module essentially creates a from/to matrix for the distances between stations. These are the traveling distances for the *transporters* not the entities and should include both loaded and empty moves. For  $n$  station locations, you will have to specify at most  $n(n - 1)$  possible distances. The distances in the distance matrix do not have to be symmetrical. In addition, you do not need to specify a distance if it never occurs (when the transporter is either loaded or empty). If a distance is not given, it is assumed to be zero. If the distance is symmetric, only one pair needs to be specified.

Figure 9.13 illustrates the general case of using a transporter and how it relates to the Arena™ modules. Since transporters are used to move entities between two stations, two stations have been indicated in the figure. At the first station, the entity performs the standard



**Figure 9.13** Activity diagram for general transporter case.

SEIZE, DELAY, and RELEASE logic. Then, the entity attempts to allocate a unit of the transporter via the REQUEST module. Notice that there is a queue for waiting for the transporter to arrive. The REQUEST also causes the transporter to move to the entity's location, as indicated with the travel time to origin activity. Then, a loading activity can occur. After the loading activity has completed, the entity experiences the delay for the transport to the desired location. As can be seen in the figure, the LEAVE module facilitates this modeling. Once the entity arrives at its destination station, there can be a delay for unloading the transfer devise and then the transporter is freed. As shown in the figure, conceptually, the HALT and ACTIVATE modules affect whether or not the transporter is available to be requested/allocated.

The activity diagram in Figure 9.13 represents very well how the test and repair system will need to operate with transporters. The next section discusses how to use transporters to model the dedicated transport workers in the test and repair shop.

### 9.3.1 Test and Repair Shop with Workers as Transporters

The results of the constrained resource analysis indicated that modeling constrained transfer for the parts in the test and repair system can have a significant effect on the system's ability to meet the contract requirements. This may be due to having the workers share the roles of tending the machines and transporting the parts. The following example investigates whether or not a set of dedicated workers would make sense for this system. In particular, the number of workers to dedicate to the transport task needs to be determined. Since there is a lot of walking involved, the model needs to be more precise in the physical modeling of the situation. This could also allow different layout configurations to be simulated if the relocation of the stations would make a difference in the efficiency of the system.

To model this situation using transporters, the distance between the stations and a way to model the velocity of transport are required. Since the workers have a natural variability in the speed of their walking, a model for human walking speed is needed. Based on some time study data, the velocity of a worker walking in the facility has been determined to be distributed according to a triangular distribution with a minimum of 22.86, a mode of 45.72, and a maximum of 52.5, all in meters per minute. Since this distribution will be used in many locations in the model, an EXPRESSION should be defined, called *eWalkingTimeCDF*, which will be equal to  $\text{TRIA}(22.86, 45.72, 52.5)$ .

Based on measuring the distance between the stations, the approximate distance between the stations has been determined as given in Table 9.3. Recall that both the loaded and unloaded distances for the transporter must be specified. For example, even though no parts are routed from repair to diagnostics, the distance from repair to diagnostics must be given

**TABLE 9.3 Transporter Distances between Stations**

		Destination			
		Diagnostics	Test 1	Test 2	Test 3
Station		Repair			
Origin	Diagnostics	—	40	70	90
	Test 1	43	—	10	60
	Test 2	70	15	—	65
	Test 3	90	80	60	—
	Repair	110	85	25	30

**Stations**

	Beginning Station	Ending Station	Distance
1	DiagnosticsStation	TestStation1	40
2	DiagnosticsStation	TestStation2	70
3	DiagnosticsStation	TestStation3	90
4	DiagnosticsStation	RepairStation	100
5	TestStation1	DiagnosticsStation	43
6	TestStation1	TestStation2	10
7	TestStation1	TestStation3	60
8	TestStation1	RepairStation	80
9	TestStation2	DiagnosticsStation	70
10	TestStation2	TestStation1	15
11	TestStation2	TestStation3	65
12	TestStation2	RepairStation	20
13	TestStation3	DiagnosticsStation	90
14	TestStation3	TestStation1	80
15	TestStation3	TestStation2	60
16	TestStation3	RepairStation	25
17	RepairStation	DiagnosticsStation	110
18	RepairStation	TestStation1	85
19	RepairStation	TestStation2	25
20	RepairStation	TestStation3	30

**Distance - Advanced Transfer**

Name	Stations	
1	TransporterDistances	20 rows

Figure 9.14 DISTANCE module.

because the worker (transporter) may be at the repair station when something needs to be moved from the diagnostic station. Thus, the worker must walk from the repair station to the diagnostic station (unloaded) in order to pick up the part. Notice also that the distances do not have to be symmetric (i.e., the distance from test 1 to test 2 does not have to be the same as the distance from test 2 to test 1).

Starting with the finished model for the resource-constrained example (without the animation), you can make small changes in order to utilize transporters. The first step is to define the distances. Figure 9.14 shows the DISTANCE module from the Advanced Transfer panel. The spreadsheet view allows an easier specification for the distances as shown in the figure. Multiple distance sets can be defined and used within the same model. The distance set must be given a name so that the name can be referenced by the TRANSPORTER module. Now, the transporters for the model can be defined using the TRANSPORTER module.

The TRANSPORTER module, see Figure 9.15, allows the transporter to be given a name and various other attributes. The TRANSPORTER module defines a fleet of identical mobile resources (e.g., vehicles). You can specify the number of units of the transporter. In this example, there will be three workers (all identical) that can transport the parts. The type of transporter in this case is Free Path and the Distance Set has been specified with the name used in the appropriate distance set from the DISTANCE module. The velocity within the TRANSPORTER module must be a real number (it cannot be an expression). The default value of the velocity will be used in the dialog because the velocity will be specified for each movement of the transporter within the model. The initial position specifies where the transporter will be located at the beginning of the replication. In this case, the workers will start active at the diagnostics station.

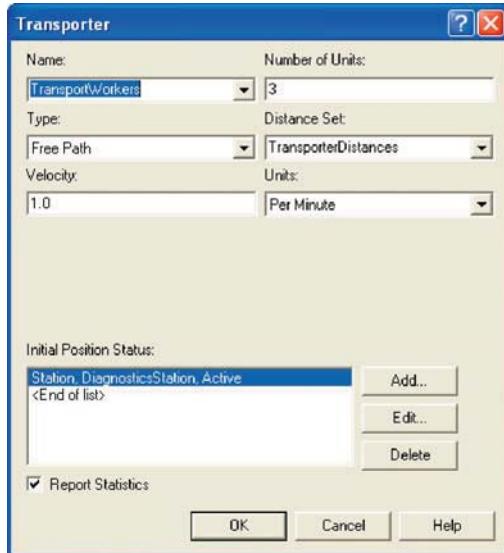


Figure 9.15 TRANSPORTER module.

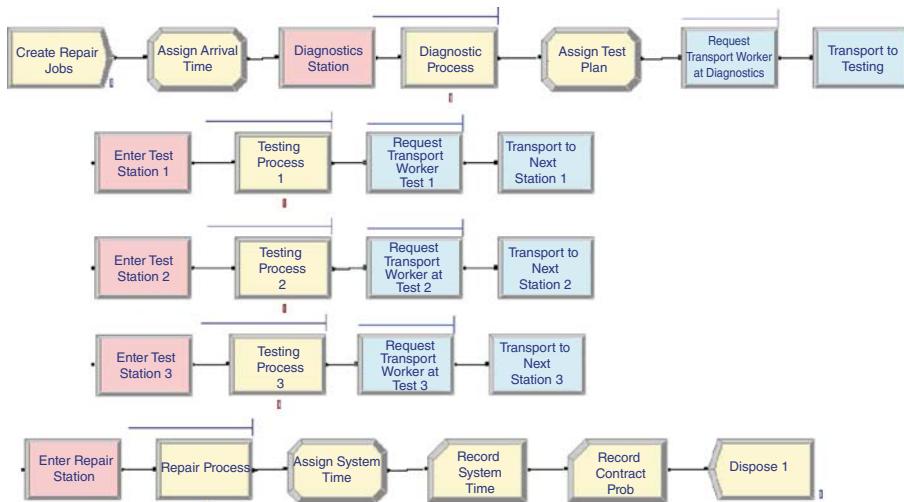
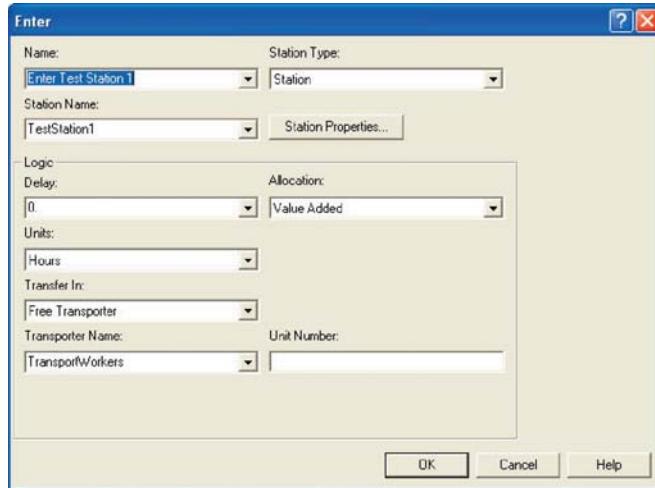


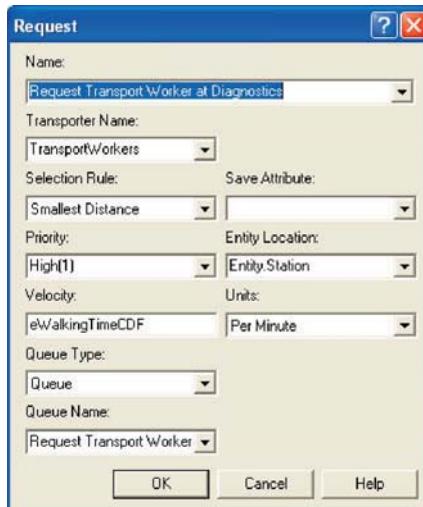
Figure 9.16 Overall test and repair model with transporters.

Now that the data modules are defined, let us take a look at the overall model. Figure 9.16 shows the overall test and repair model after the modifications needed to use transporters. If you have the model open, you should notice the new blue-colored modules. These are from the Advanced Transfer panel and have replaced the LEAVE modules from the previous model.

The other change from the previous model involves the updating of the ENTER modules. The ENTER module change is very straightforward. You just need to specify that the type of transfer in option is Free Transporter and indicate what transporter to free. Figure 9.17



**Figure 9.17** ENTER module with free transporter option.



**Figure 9.18** REQUEST module.

illustrates the changes to the ENTER module for entering test station 1. By default, the transporter unit that the entity currently has is freed.

The changes to the LEAVE module are not as simple. Because a random velocity value is needed whenever the worker begins moving, the LEAVE module cannot be used. The LEAVE module relies on the default velocity for the transporter as defined in the TRANSPORTER module. Since the default velocity in the TRANSPORTER module must be a real number (not an expression), the REQUEST and TRANSPORT modules are used in this example.

Figure 9.18 shows the REQUEST module for the example. In the REQUEST module, you must specify which transporter to request and give the selection rule. The selection

rules are similar in concept to the selection rules for resources. The transporter selection rules are

**Cyclical** Cyclic priority selects the first available transporter beginning with the successor of the last transporter allocated (cycles through the transporters).

**Largest Distance** Select the available transporter farthest from the requesting station.

**Preferred Order** Select the available transporter unit which has the lowest unit number.

**Random** Select randomly from the available transporter units.

**Smallest Distance** Select the nearest available transporter.

**Specific Member** A specific unit of the transporter.

**ER(Index)** Select based on rule number Index defined in experiment frame.

**UR(Index)** Select transporter index UR, where UR is computed in a user-coded function UR(LENL,NUR).

The selection rule is only used if one or more units of the transporter are available to be allocated. An entity arrives at the REQUEST module and requests the transporter according to the specified selection rule. If a unit of the transporter is available (active and idle), the transporter is allocated to the entity, thereby making the transporter busy (and allocated). The selected transporter can be saved in the save attribute. The travel time is calculated from the transporter's present station to the station of the requesting entity and imposes a time delay representing the travel time using the velocity argument. If a transporter is not available, the entity waits in queue. This module gets the transporter to the entity. Now you have to use the transporter to transport the entity using the TRANSPORT module.

Figure 9.19 shows the TRANSPORT module for the test and repair example. The TRANSPORT module is very similar to the ROUTE module, except the user specifies the transporter to use by name (and/or unit number). In most cases, this will be the same transporter as allocated in the REQUEST module. Then, the user can specify the type of destination (By Sequence, attribute, station, or expression). In this case, the By Sequence option should be used. In addition, the velocity of the transport can be specified. The Guided Tran Destination Type field allows the user to send the transporter to a location

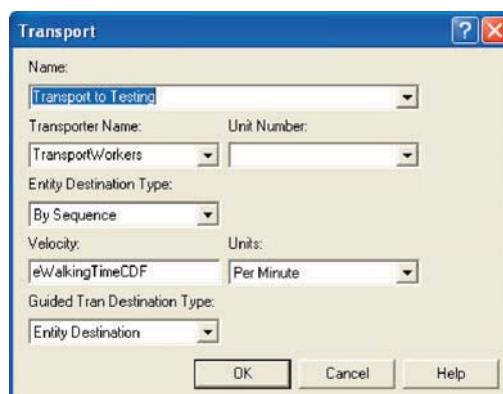


Figure 9.19 TRANSPORT module.

Transporter		
Usage		
Number Busy	Average	Half Width
TransportWorkers	0.4660	0.00
Number Scheduled	Average	Half Width
TransportWorkers	3.0000	0.00
Utilization	Average	Half Width
TransportWorkers	0.1553	0.00

Figure 9.20 Transporter statistics.

different from that specified by the entity's destination. This is useful in guided path transporters to send the transporter through specific intersections on its way to the final destination. This option will not be used in this text.

After making these changes, you can run the model with the default animation and find out how the system operates with the transporters. Recall that in Figure 9.15, there was a little check-box to indicate whether or not to collect statistics on the transporter. If this is checked, and the Transporters check-box is checked on the Project Parameters tab on the Run Setup dialog, the number busy and utilization of the transporters will be reported automatically.

As can be seen in Figure 9.20, the utilization of the three transporters is very low. Less than three workers will probably be needed for the transport task. The reader is asked to explore this issue as an exercise.

### 9.3.2 Animating Transporters

A significant advantage of using transporters in this situation rather than resource-constrained transfer is the animation that is available with transporters. This section converts the previous animation so that it uses transporters. The basic steps in converting the model are as follows:

1. Cut and paste the animation from the file *RepairShopResourceConstrainedTransferWithAnimation.doe* to the file *RepairShopWithTransportersNoAnimation.doe* and rename it.
2. Select and delete each route connector and station marker from the previous animation.
3. Delete each animation queue from the flowchart modules. Redefine each animation queue with the appropriate name within the copied animation.
4. Use the Add Distance button on the Animate Transfer toolbar to place station markers and distance connectors for every from/to combination within the distance set as shown in Figure 9.21. For every station marker, make sure that the "Parking" check-box is checked. This will indicate that the transporter will be shown at that station when it is idle (parked).

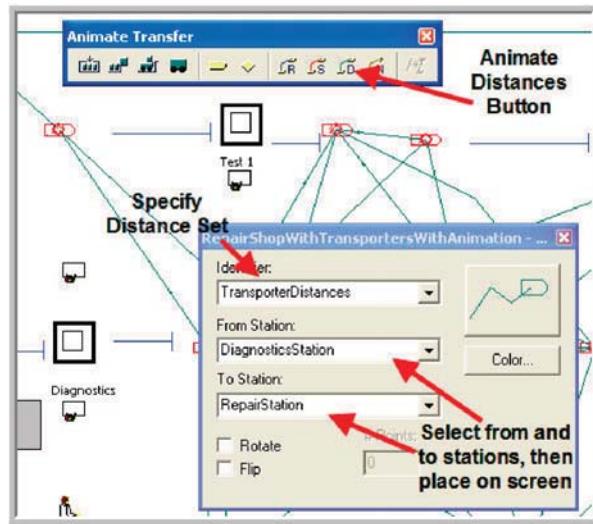


Figure 9.21 Placing the distance animation elements.

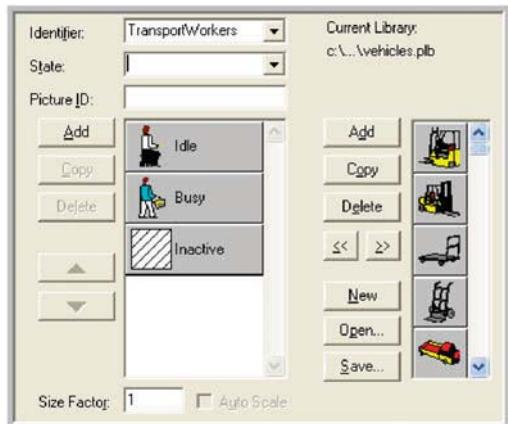


Figure 9.22 Defining the transporter picture.

5. Add a variable animation to show the current number of busy transporters. This can be easily done using the expression builder to find the transporter variables ( $NT(TransportWorkers)$ ). To learn more about the variables associated with transporters look up “Transporter Variables” within Arena’s help system.
6. Use the transporter button in the Animate Transfer toolbar to define a transporter picture. The process is essentially the same as that for defining a resource animation picture as shown in Figure 9.22.

Now the animation involving the transporters is complete. If you run the model with animation turned on, you will see that there is only about 1 unit of the transporter busy at any time. Also, you will see that the idle transporters become parked after their transport when no other transport requests are pending. They will remain at the station where they

last dropped off a part until a new transport request comes in. In the animation, a picture that showed a person sitting idle and walking/carrying something to show busy was used. Another way to show that the transporter is carrying something is to enable the *ride point* for the transporter. To place a ride point in the busy transporter picture, select the Ride Point option from the Object menu after double-clicking on the busy picture in the transporter animation dialog. If no ride point is defined, the entity picture will not appear.

In this model, the worker stays at the location where they drop off the part if no part requires transport. Instead, the worker could go to a common staging area (e.g., in the middle of the shop) to wait for the next transport request. The trick to accomplishing this is not to use the ENTER module. The basic idea would be implemented as follows:

```

STATION
DECIDE
  IF requests are waiting THEN
    FREE the transporter
    continue normal station processing
  ELSE
    SEPARATE 1 duplicate into station
    MOVE to staging area
    FREE the transporter
  ENDIF
END DECIDE

```

Notice that in the case of no waiting requests, a duplicate of the entity is used to continue the control of the transporter to move it to the staging area. The reader is asked to implement this idea as an extension to this problem in the exercises. The notion of having an entity control or “drive” the transporter around is very useful in modeling systems that have complicated transporter allocation rules or systems that naturally have a driver, for example, bus systems.

A DISTANCE module allows the physical transport aspects of systems to be modeled. An important aspect of this type of modeling is the movement of entities through space. The conveyor constructs allow for a finer representation of the usage of space between stations. That topic is taken up in the next section.

## 9.4 MODELING SYSTEMS WITH CONVEYORS

A conveyor is a track, belt, or some other device that provides movement over a fixed path. Typically, conveyors are used to transport items that have high volumes over short to medium range distances. The speeds of conveyance typically range from 20 to 80 feet per minute to as fast as 500 feet per minute. Conveyors can be gravity based or powered. The modeling of conveyors can be roughly classified as follows:

**Accumulating** Items on the conveyor continue to move forward when there is a blockage on the conveyor.

**Nonaccumulating** Items on the conveyor stop when the conveyor stops

**Fixed Spacing** Items on the conveyor have a fixed space between them or the items ride in a bucket or bin.

**Random Spacing** Items are placed on the conveyor in no particular position and take up space on the conveyor.

You have experienced conveyors in some form. For example, the belt conveyor at a grocery store is like an accumulating conveyor. An escalator is like a fixed spaced nonaccumulating conveyor (the steps are like a bucket for the person to ride in. People movers in airports are like nonaccumulating, random spacing conveyors. When designing systems with conveyors, there are a number of performance measures to consider.

**Throughput Capacity** The number of loads processed per time.

**Delivery Time** The time taken to move the item from origin to destination.

**Queue Lengths** The queues to get on the conveyor and for blockages when the conveyor accumulates.

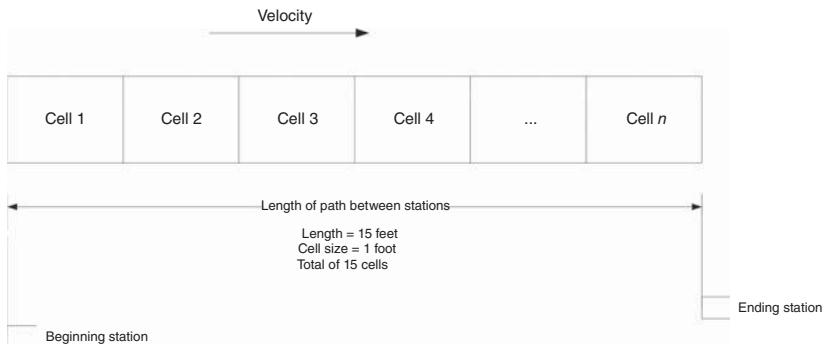
**Number of Carriers Used or the Space Utilized** The number of spaces on the conveyor used.

The modeling of conveyors does not necessarily require specialized simulation constructs. For example, gravity-based conveyors can be modeled as a PROCESS or a DELAY with a deterministic delay. The delay is set to model the time that it takes the entity to fall or slide from one location to another. One simple way to model multiple items moving on a conveyor is to use a resource to model the front of the conveyor with a small delay to load the entity on the conveyor. This allows for spacing between the items on the conveyor. Once on the conveyor, the entity releases the front of the conveyor and delays for its move time to the end of the conveyor. At the end of the conveyor, there could be another resource and a delay to unload the conveyor. As long as the delays are deterministic, the entities will not pass each other on the conveyor. If you are not interested in modeling the space taken by the conveyor, then this type of modeling is very reasonable. Henriksen and Schriber [1986] discussed a number of ways to approach the simulation modeling of conveyors. When space becomes an important element of the modeling, then the simulation language constructs for conveyors become useful. For example, actually, if there is a conveyor between two stations and the space allocated for the length of the conveyor is important to the system operation, you might want to use conveyor-related modules.

In Arena<sup>TM</sup>, a conveyor is a material-handling device for transferring or moving entities along a predetermined path having fixed predefined loading and discharge points. Each entity to be conveyed must wait for sufficient space on the conveyor before it can gain entry and begin its transfer. In essence, simulation modeling constructs for conveyors model the travel path via a mapping of space/distance to resources. Space along the path is divided up into units of resources called cells. The conveyor is then essentially a set of moving cells of equal length. Whenever an entity reaches a conveyor entry point, it must wait for a predefined amount of unoccupied and available consecutive cells in order to get on the conveyor.

Figure 9.23 illustrates the idea of modeling a conveyor as a set of contiguous cells representing the space on the conveyor. One way to think of this is like an escalator with each cell being a step.

In the figure, if each cell represents 1 foot and the total length of the conveyor is 15 feet, then there will be 15 cells. The cell size of the conveyor is the smallest portion of a conveyor that an entity can occupy. In modeling with conveyors, the size of the entity also matters. For example, think of people riding an escalator. Some people have a suitcase and



**Figure 9.23** A conveyor conceptualized as a set of contiguous cells.

take up two steps and others do not have a suitcase and only take up one step. An entity must acquire enough contiguous cells to hold their physical size in order to be conveyed.

The following are the key modules for modeling the use of conveyors.

**ACCESS** When an entity enters an ACCESS module, it will wait until the appropriate amount of contiguous cells are available at the access point. Once the entity has control of the cells, it can then be conveyed from its current station to a station associated with the conveyor. This is similar in concept to the ALLOCATE module for transporters.

**CONVEY** The CONVEY module causes the entity to move from its origin station to its next station. The time to move is based on the velocity of the conveyor and the distance between the stations. This is similar in concept to the MOVE module for transporters.

**EXIT** The EXIT module causes the entity to release the cells that it holds on the conveyor. If another entity is waiting in queue for the conveyor at the same station where the cells are released, the waiting entity will then access the conveyor. This is like releasing a resource or freeing a transporter.

**START** The START module changes the status of the conveyor to active. Entities may reside on the conveyor while it is stopped. These entities will maintain their respective positions on the conveyor once it is started. The entity using the START module does not have to be on the conveyor.

**STOP** The STOP module changes the status of the conveyor to inactive. The conveyor will stop immediately, regardless of the number of entities on the conveyors. This is useful for modeling conveyor failures or blockages on the conveyor. The entity using the STOP module does not have to be on the conveyor.

**CONVEYOR** This data module defines whether or not the conveyor is an accumulating or nonaccumulating conveyor, the cell size (in physical units, e.g., feet), and the velocity. A conveyor consists of a sequence of segments as defined by the SEGMENT module. The sum of the distances among the stations on the conveyor (specified by SEGMENT module) must be divisible by the cell size. If not, an error will occur when the model is checked.

**SEGMENT** This data module defines the segment network that makes up a conveyor. Each segment is a directed link between two stations forming a network. The

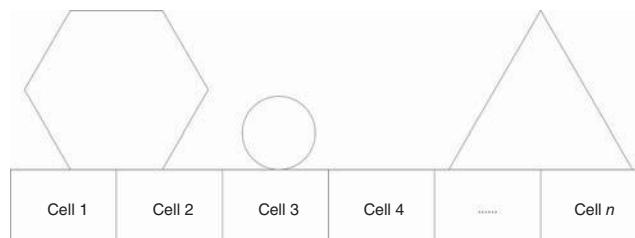
conveyor path is defined by a beginning station and a set of next station-distance pairs. The beginning station of the segment is the beginning station number or name, and the next station defines the next station name that is a length of distance units from the previously defined station. The length of the segment must be specified in integer distance units (feet, meters, etc.). No station should be repeated within the segment network unless the conveyor is a loop conveyor. In that case, the last ending station should be the same as the beginning station for the segment network.

As was the case for transporters, the ENTER and LEAVE modules of the advanced transfer template will also provide some conveyor functionality. LEAVE provides for ACCESS and CONVEY. ENTER provides for EXIT. The modules work slightly different for accumulating and nonaccumulating conveyors.

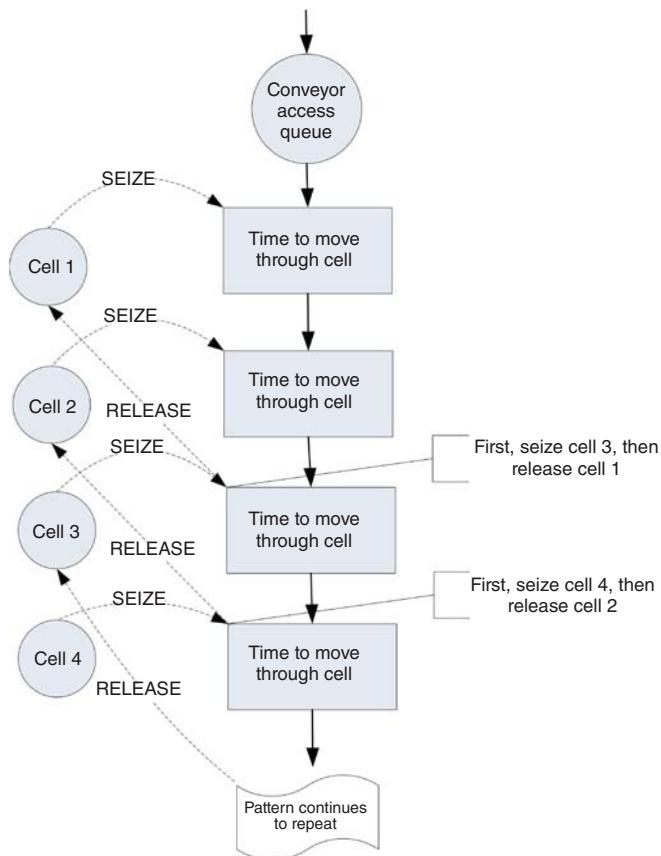
Nonaccumulating conveyors travel in a single direction, and the spacing remains the same between the entities. When an entity is placed on the conveyor, the entire conveyor is actually disengaged or stopped until instructions are given to transfer the entity to its destination. When the entity reaches its destination, the entire conveyor is again disengaged until instructions are given to remove the entity from the conveyor, at which time it is engaged or started.

As previously mentioned, the conveyor is divided into a number of cells. To get on the conveyor and begin moving, the entity must have its required number of contiguous cells. For example, in Figure 9.24, the circle needed one cell to get on the conveyor. Suppose the hexagon was trying to get on the conveyor. As cell 2 became available, the hexagon would seize it. Then it would wait for the next cell (cell 1) to become available and seize it. After having both required cells, it is “on” the conveyor. But moving on a conveyor is a bit more continuous than this. Conceptually, you can think of the hexagon, starting to move on the conveyor when it gets cell 2. When one cell moves, all cells must move in lock step. Since entities are mapped on to the cells by their size, when a cell moves, the entity moves. Reversing this analogy becomes useful. Suppose the cells are fixed, but the entities can move. In order for an entity to “take a step,” it needs the next cell. As it crosses over to the next cell, it releases the previous cell that it occupied. It is as if the entity’s “step size” is one cell at a time.

Figure 9.25 illustrates an approximate activity flow diagram for an entity using a nonaccumulating conveyor. Suppose the entity size is 2 feet and each cell is 1 foot on the conveyor. There is a queue for the entities waiting to access the conveyor. The cells of the conveyor act as resources modeling the space on the conveyor. The movements of the entity through the space of each cell are activities. For the 2-feet entity, it first seizes the first cell and then moves through the distance of the cell. After the movement, it seizes the next cell and moves through its length. Because it is 2-feet (cells) long, it seizes the third cell and then releases



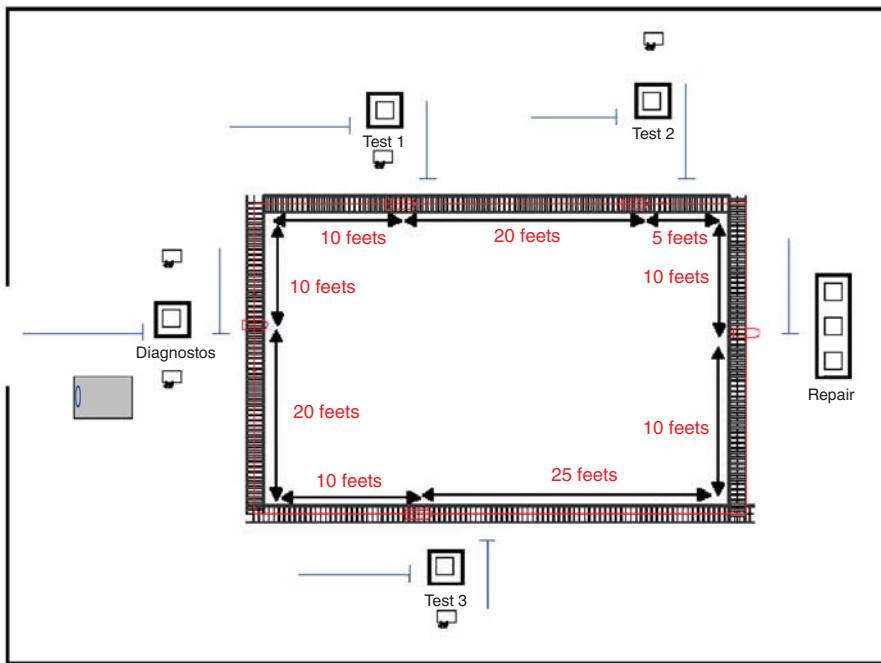
**Figure 9.24** Different entity sizes on a conveyor.



**Figure 9.25** Activity diagram for nonaccumulating conveyor.

the first cell before delaying for the move time for the third cell. As cells become released, other entities can seize them and continue their movement. The repeated pattern of overlapping SEIZE and RELEASE modules that underlie conveyor modeling clearly depends upon the number of cells and the size of the entity. Thus, the activity diagram would actually vary by the size of the entity (number of cells required).

The larger the number of cells to model a segment of a conveyor, the more slowly the model will execute; however, a larger number of cells allows for a more “continuous” representation of how entities actually exit and leave the conveyor. Larger cell sizes force entities to delay longer to move and thus waiting entities must delay for available space longer. A smaller mapping of cells to distance allows then entity to “creep” onto the conveyor in a more continuous manner. For example, suppose the conveyor was 6-feet long and space was modeled in inches, that is, the length of the conveyor is 72 inches. Now suppose the entity required 1 foot of space while on the conveyor. If the conveyor is modeled with 72 cells, the entity starts to get on the conveyor after only 1 inch (cell) and requires 12 cells (inches) when riding on the conveyor. If the mapping of distance was 1 cell equals 1 foot, the entity would have to wait until it get a whole cell of 1 foot before it moved onto the conveyor.



**Figure 9.26** Test and repair shop with conveyors.

Now you are ready to put these concepts into action.

#### 9.4.1 Test and Repair Shop with Conveyors

For simplicity, the test and repair shop will be used for illustrating the use of conveyors. Figure 9.26 shows the test and repair shop with the conveyors and the distance between the stations. From the figure, the distances from each station going in clockwise order are as follows:

- Diagnostic station to test station 1, 20 feet
- Test station 1 to test station 2, 20 feet
- Test station 2 to repair, 15 feet
- Repair to test station 3, 45 feet
- Test station 3 to diagnostics, 30 feet.

Assume that the conveyor's velocity is 10 feet per minute. These figures have been provided in this example; however, in modeling a real system, you will have to tabulate this information. To illustrate the use of entity sizes, assume the following concerning the parts following the four test plans:

- Test plan 1 and 2 parts require 1 foot of space while riding on the conveyor.
- Test plan 3 and 4 parts require 2 feet of space while riding on the conveyor.

	Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics
1	vContractLimit						
2	vMTBA						
3	vSizes	4					

Double-click here to add a new row.

Initial Values	
1	1
2	1
3	2
4	2

Figure 9.27 Array for part sizes.

Segment - Advanced Transfer			
	Name	Beginning Station	Next Stations
1	ShopConveyor.Segment	DiagnosticsStation	5 rows

Double-click here to add a new row.

Next Stations		
	Next Station	Length
1	TestStation1	20
2	TestStation2	20
3	RepairStation	15
4	TestStation3	45
5	DiagnosticsStation	30

Double-click here to add a new row.

Figure 9.28 SEGMENT module for test and repair example.

The conveyor logic will first be implemented without animation. Then, the animation for the conveyors will be added. To convert the previous model so that it uses conveyors, the model *RepairShopWithTransportersNoAnimation.doe* should be changed as follows:

1. Delete the TRANSPORTER and DISTANCE modules defined for using the transporter.
2. Delete the REQUEST and TRANSPORT modules for leaving the stations and replace them with LEAVE modules. The LEAVE modules will be filled out in a moment.
3. Add a variable array called *vSizes()* of dimension 4 to hold the sizes of the parts following each of the test plans as shown in Figure 9.27.
4. Save your file. The completed model can be found in the file *RepairShopWithConveyorsWithNoAnimation.doe*.

Now, you are ready to define the conveyor and its segments. You will first define the segments for the conveyor, then the conveyor itself. Figure 9.28 shows the segments to be used on the conveyor. First, give the segment a name and then specify the station that represents the *beginning station*. The beginning station is the station associated with the first segment of the conveyor. Since this is a loop conveyor, the beginning station is arbitrary. The diagnostic station is used here since this is where the parts enter the system.

Then, the next station and the distance to the next station are given for each part of the conveyor. Notice that the distances have been added as shown in Figure 9.26. The lengths

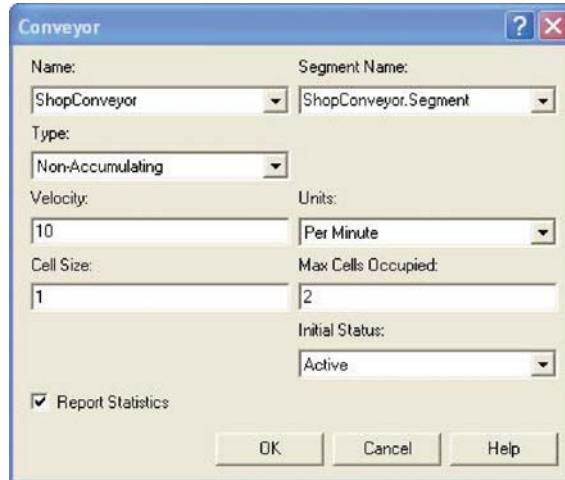


Figure 9.29 CONVEYOR module.

specified in Figure 9.28 are in the actual physical distances (e.g., feet). The mapping to cells occurs in the CONVEYOR module.

Figure 9.29 shows the CONVEYOR module. To associate the conveyor with a segment, use the Segment Name drop down box. The next decision is the type of conveyor. In this case, a nonaccumulating conveyor should be used.

The velocity of the conveyor is 10 feet per minute. Now, the cell size and the maximum cells occupied text-boxes must be completed. The cell size is the most important decision. Thinking back to Figure 9.25, it should be clear that space must be represented by discrete cells of a specified integral length. Thus, the total length of the conveyor must be integer valued and the cell size must result in an integer number of cells for each segment. The lengths of each distance specified in the SEGMENT module for the example are all divisible by 5. Thus, you could choose a cell size of 5. This would mean that each cell would be 5 feet in length. Since entities access the conveyor by grabbing cells, this hardly seems appropriate if the parts are 1 and 2 feet in length, respectively. In the case of a 5 feet cells size, the parts would have a lot of space between each other as they ride on the conveyor. Instead, the cell size will be specified as 1 foot. This will always work, since every number is divisible by 1. It also works well in this case, since the smallest part is 1 foot and the distances in the SEGMENT module are based on the units of feet. Now, the maximum number of cells text field is easy. This represents the size of the biggest entity in terms of cells. This should be 2 in this case since parts follow test plans 3 and 4 are 2 feet in length.

With the CONVEYOR and SEGMENT modules defined, it is an easy matter to change the ENTER and LEAVE modules to use the conveyors. The first item to address is to make sure that each created part knows its size so that the size can be used in the LEAVE module. You can do this with the “Assign Test Plan” ASSIGN module as shown in Figure 9.30.

The LEAVE modules must be updated to use the conveyor to transfer out and to indicate the number of cells required by the entity. This is done in Figure 9.31 using the entity’s *mySize* attribute. Each of the other LEAVE modules for the other stations should be updated in a similar manner. Now, you must update the ENTER module so that the entity can exit the conveyor after arriving to the desired station.

	Type	Attribute Name	New Value
1	Attribute	myTestPlan	eTestPlanCDF
2	Attribute	Entity.Sequence	TestPlanSequences(myTestPlan)
3	Attribute	Entity.Picture	MEMBER(PartPictures,myTestPlan)
4	Attribute	mySize	vSizes(myTestPlan)

Double-click here to add a new row.

Figure 9.30 Assigning the size of the parts.

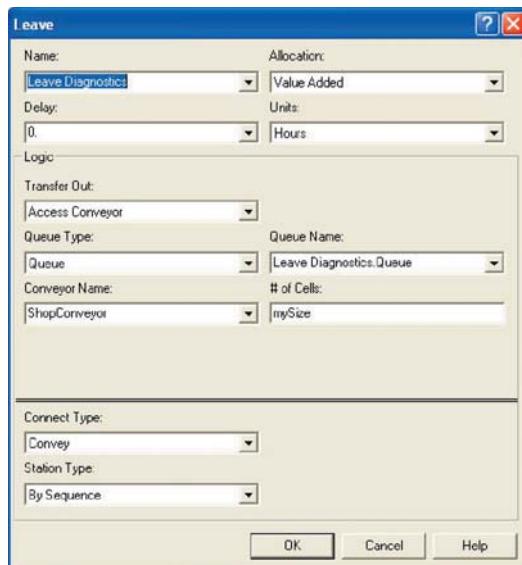


Figure 9.31 Example LEAVE module for test and repair example.

In Figure 9.32, the Exit Conveyor option is used when transferring into the Test 1 Station. In both the ENTER and LEAVE modules, the option of delaying the entity is available. This can be used to represent a loading or an unloading time for the conveyor.

With the conveyor statistics check-box clicked, the conveyor statistics will appear on the summary reports. The utilization is calculated for the conveyor, regardless of conveyor type. This utilization represents the length of entities on the conveyor in comparison to the length of the entire conveyor over the simulation run. Thus, conveyor utilization is in terms of the space utilized. In addition, there is very little queuing for getting on the conveyor. The statistics for the conveyor, see Figure 9.33, in this problem indicate that the conveyor is not highly utilized; however, the chance of meeting the contract limit is up to 80%. The volumes in this example may make it difficult to justify the use of conveyors for the test and repair shop. Ultimately, the decision would come down to a cost/benefit analysis.

#### 9.4.2 Animating Conveyors

To augment the model with animation, the Animate Transfer toolbar can be used. To visibly discern the plan that an entity is following within the animation, you can define a picture

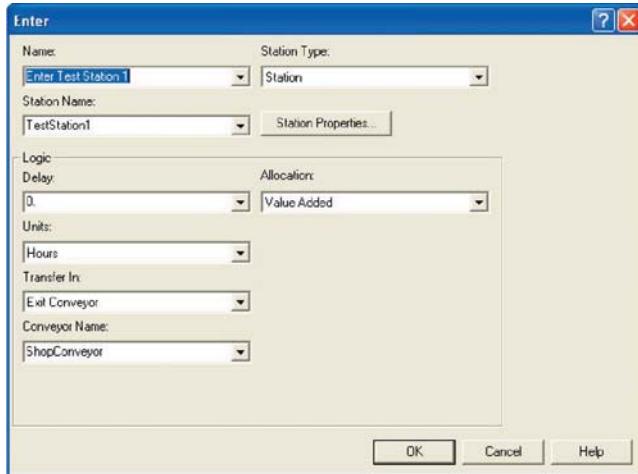


Figure 9.32 ENTER module for Test Station 1.

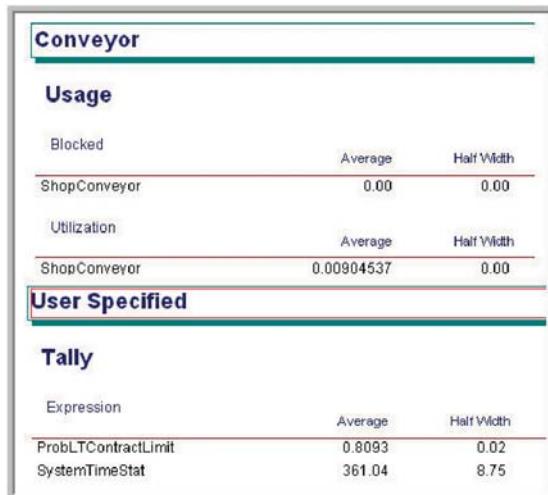
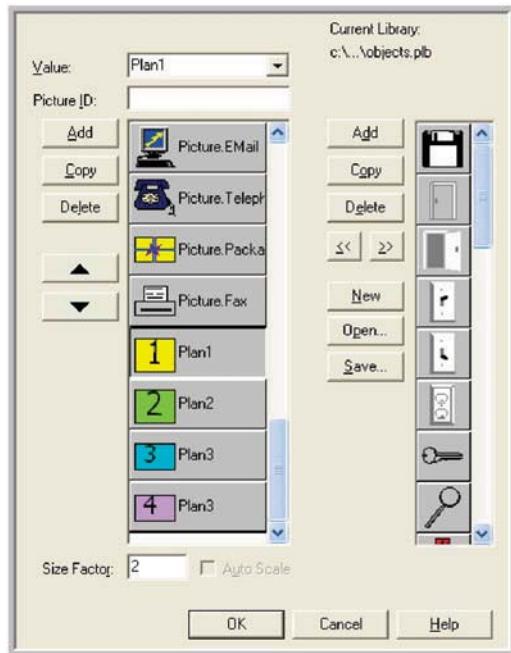


Figure 9.33 Statistics for test and repair with conveyors.

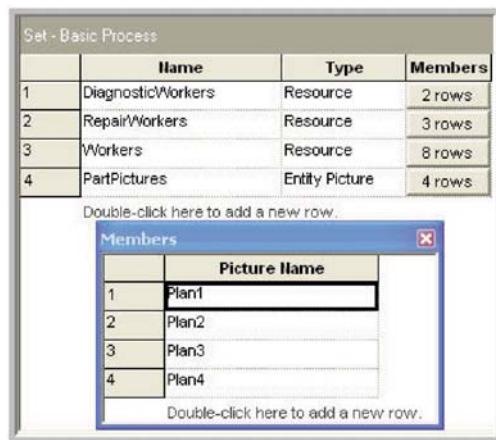
set, with appropriately drawn pictures for each type of plan. This is illustrated in Figures 9.34 and 9.35. The assignment to the Entity.Picture attribute is shown in Figure 9.30.

To update the previous transporter animation, perform the following steps.

1. Copy and paste the animation elements from the file *RepairShopWithTransporters-WithAnimation.doe* to your own model file. The completed Arena™ model is called *RepairShopWithConveyorsWithAnimation.doe*.
2. Delete the transporter picture, station markers, and distance connectors.
3. Using the Animate Transfer toolbar's Segment button, make station markers and segment connectors to represent the conveyors as shown in Figure 9.26. The segment animation dialog is shown in Figure 9.36. It works very much like the route and distance animation dialog.



**Figure 9.34** Entity pictures for test plans.



**Figure 9.35** Picture set for test plans.

After the segments have been laid down, you might want to place some lines to represent the conveyors. This was done in Figure 9.36 where the roller pattern from the line patterns button on the drawing toolbar was also used. That's it! Animating conveyors is very easy. If you run the model with the animation on, you will see that the entities move along the conveyor segments, getting on and off at the appropriate stations. You will also clearly see that for the majority of time, the conveyor does not have very many parts. This section concentrated on nonaccumulating conveyors. The next section describes how accumulating conveyors operate and discusses a number of other conveyor modeling issues.

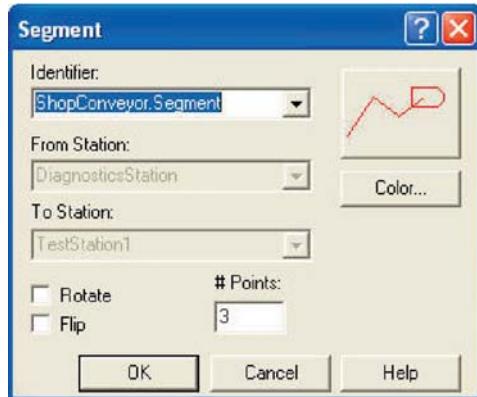


Figure 9.36 Segment animation dialog.

#### 9.4.3 Miscellaneous Issues in Conveyor Modeling

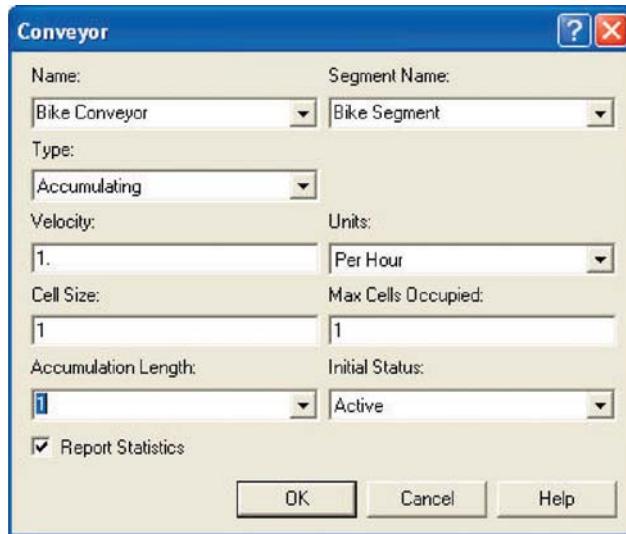
This section discusses accumulating conveyors, how to model merging conveyors, diverging conveyors, how to allow processing to occur on the entity while it is still on the conveyor, and recirculation conveyors.

**9.4.3.1 Accumulating Conveyors** An accumulating conveyor can be thought of as always running. When an entity stops moving on the conveyor (e.g., to unload), other entities are still allowed on the conveyor since the conveyor continues moving. When entities on the conveyor meet up with the stopped entity, they stop, and a queue accumulates behind the stopped entity until the original stopped entity is removed or transferred to its destination.

As an analogy, imagine wearing a pair of roller blades and being on a people mover in an airport. Do not ask how you got your roller blades through security! While on the people mover, you are not skating (you are standing still, resting, but still moving with the people mover). Up ahead of you, some seriously deranged simulation book author places a bar across the people mover. When you reach the bar, you grab on. Since you are on roller blades, you remain stationary at the bar, while your roller blades are going like mad underneath you.

Now imagine all the people on the mover having roller blades. The people following you will continue approaching the bar until they bump into you. Everyone will pile up behind the bar until the bar is removed. You are on an accumulating conveyor! Notice that while you were on the conveyor and there was no blockage, the spacing between the people remained the same, but that when the blockage occurred the spacing between the entities decreased until they bump up against each other. To summarize,

- Accumulating conveyors are always moving.
- If an entity stops to exit or receives processing on the conveyor, other entities behind it are blocked and begin to queue up on the conveyor.
- Entities in front of the blockage continue moving.
- When a blockage ends, blocked entities, may continue, but must first wait for the entities immediately in front of them to move forward.



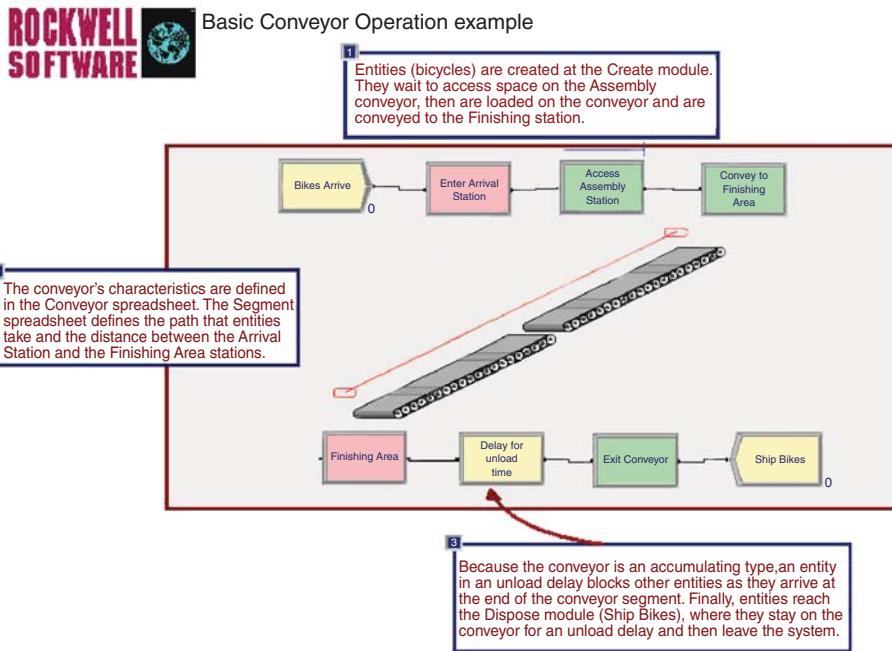
**Figure 9.37** Accumulating conveyor dialog.

In modeling accumulating conveyors, the main differences occur in the actions of the ACCESS and CONVEY modules. Instead of disengaging the conveyor as with nonaccumulating conveyors, the conveyor continues to run. ACCESS allocates the required number of cells to any waiting entities as space becomes available. Any entities that are being conveyed continue until they reach a blockage. If the blocking entity is removed or conveyed, the accumulated entities only start to move when the required number of cells becomes available.

In the ACCESS module, the number of cells still refers to the number of cells required by the entity while moving on the conveyor; however, you must indicate what the space requirements will be when the entities experience a blockage. In the CONVEYOR module, if the accumulating conveyor option is selected, the user must decide on how to fill in the Accumulation Length text field. The accumulation length specifies the amount of space required by the entities when accumulating. It does not need to be the same as the amount of space implied by the number of cells required when the entity is being conveyed. Also, it does not have to be divisible into an even number of cells. In addition, as can be seen in Figure 9.37, the accumulation length can also be an entity (user-defined) attribute. Thus, the accumulation size can vary by entity. Typically, the space implied by the number of cells required while moving will be larger than that required when accumulating. This will allow spacing between the entities when moving and allow them to get closer to each other when accumulating.

The test and repair example barely required nonaccumulating conveyors. Thus, to illustrate accumulating conveyors, a SMART file, (*Smarts101.doe*), will be used (Figure 9.38). The SMART files can be found within the Arena™ folder within your installation of Arena™.

In this example, there is 1 segment for the conveyor of length 10, which (while not specified in the model) can be assumed to be in meters. The velocity is only 1 meter per hour, the conveyor cell size is 1, and the maximum size of an entity is 1. The entities, which are bicycles, enter at the arrival station and access the conveyor, where they are conveyed to the finishing area. You can think of the bicycles as being assembled as they move along the



**Figure 9.38** Arena's Smarts101.doe model for accumulating conveyors.

conveyor. This type of system often occurs in assembly lines. Once the bicycles reach the finishing area, they experience an unload delay prior to exiting the conveyor. Because the unload delay occurs prior to exiting the conveyor, the entity's movement is blocked and an accumulation queue will occur behind the entity on the conveyor. You should run the model and convince yourself that the bicycles are accumulating.

When modeling with accumulating conveyors, additional statistics are collected on the average number of accumulating entities if the conveyors check-box is checked on the Project Parameters tab of the Run Setup dialog box. You can check out the help system to see all the statistics.

**9.4.3.2 Merging and Diverging Conveyors** In conveyor modeling, a common situation involves one or more conveyors feeding another conveyor. For example, in a distribution center, there may be conveyors associated with each unloading dock, which are then attached to a main conveyor that takes the products to a storage area. To model this situation within Arena™, the feeding conveyors and the main conveyor can be modeled with different CONVEYOR modules and separate SEGMENT definitions. In this distribution center example, an item accesses the unloading dock's conveyor rides the length of the conveyor and then attempts to get on the main conveyor. After it has accessed the main conveyor, the entity exits its current unloading dock conveyor.

Arena™ has two SMART files that illustrate these concepts. Let us first take a look at *Smarts107.doe*. Figure 9.39 illustrates the overall model. There are two conveyor lines for area 1 and area 2, respectively. When the entities arrive to their area, an attribute called *LineNum* is assigned to indicate where it came from. The entities access the appropriate conveyor and then are conveyed to the sorting area. The sorting area in this example is

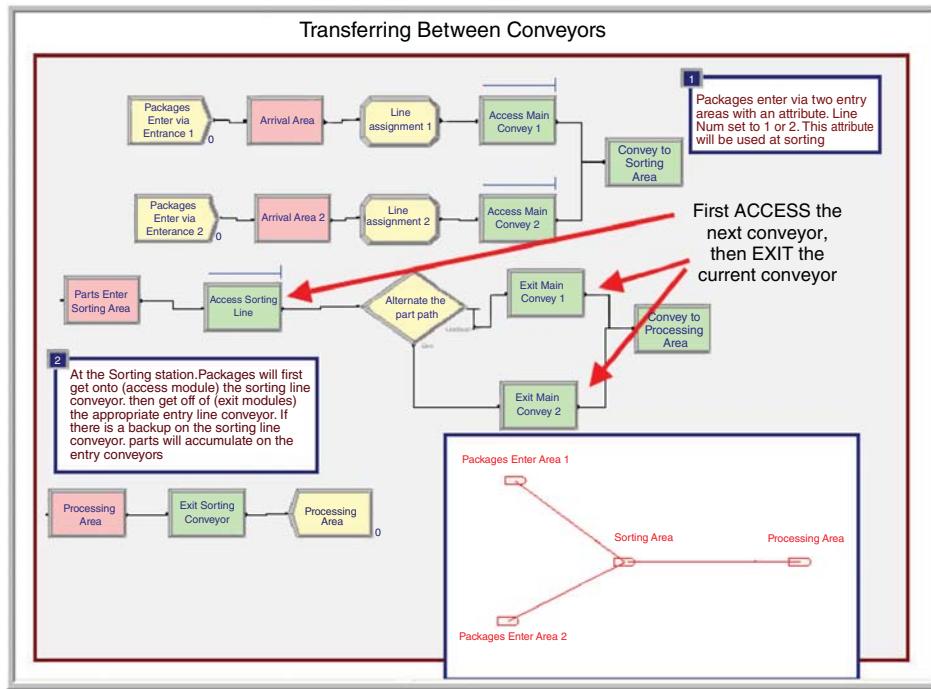
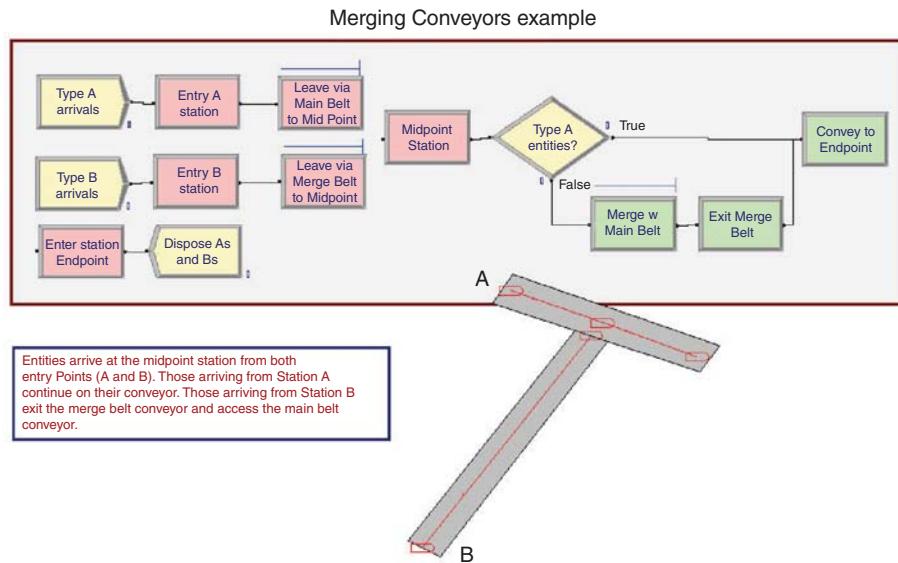


Figure 9.39 Transferring between conveyors.

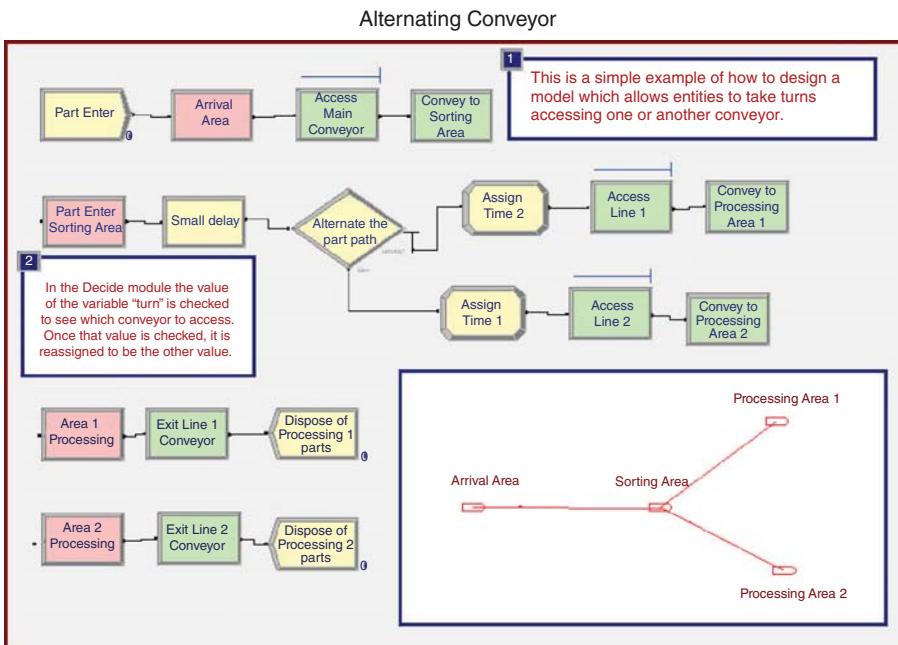
the station where the main conveyor begins. Once the parts arrive at the sorting station, they first access the conveyor to the processing area. Then, the entity exits its current conveyor. The DECIDE module is used to select the correct conveyor to exit based on the *LineNum* attribute. By first accessing the next conveyor, the entity may have to wait on its current conveyor. Because of this, it is very natural to model this situation with accumulating conveyors. The feeder conveyors will back up as entities try to get on the main conveyor.

A similar example involves the merging of one conveyor onto another conveyor as in SMART file *Smarts110.doe* (Figure 9.40). In this example, there are two conveyors. The main conveyor has two segment lengths associated with it. For example, the main conveyor goes from Entry A Station to the Midpoint Station to the EndPoint station, with lengths 5 feet, respectively. The second conveyor is a single conveyor from the Entry B station to the MidPoint station with a length of 10 feet. Notice that the MidPoint station is associated with segments related to the two conveyors. In other words, they share a common station. The entities arrive at both the Entry A and Entry B stations. Entities that arrive at the Entry A station first convey to the MidPoint station and then immediately convey to the EndPoint station. They do not exit the conveyor at the MidPoint station. Entities from the Entry B station are conveyed to the MidPoint station. Once at the MidPoint station, they first ACCESS the main conveyor and then EXIT their conveyor. This causes entities coming from Entry B to potentially wait for space on the main conveyor. This basic idea can be expanded to any number of feeder conveyors along a longer main conveyor.

Diverging conveyors are often used in system that sort items. The items come in on one main conveyor and are transferred to any number of other conveyors based on their

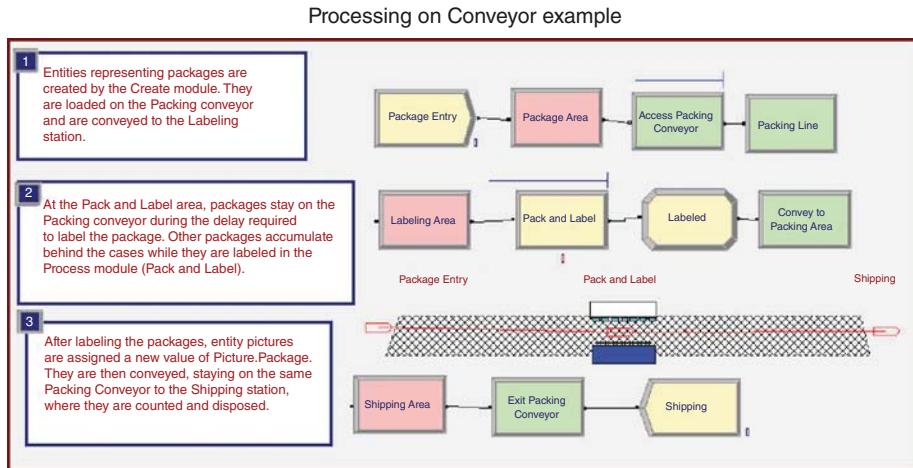


**Figure 9.40** One conveyor merging into another.



**Figure 9.41** Simple alternating conveyor.

attributes (e.g., destination). Arena's *Smarts108.doe* file illustrates the basic concepts behind diverging conveyors. In this example, as shown in Figure 9.41, the parts arrive to the arrival area and access a conveyor to the sorting area. Once an entity reaches the sorting station, the entity first exits the incoming conveyor and then there is a small delay for the sorting. The module labeled "Parts Enter Sorting Area" is an ENTER module with



**Figure 9.42** Processing while on a conveyor.

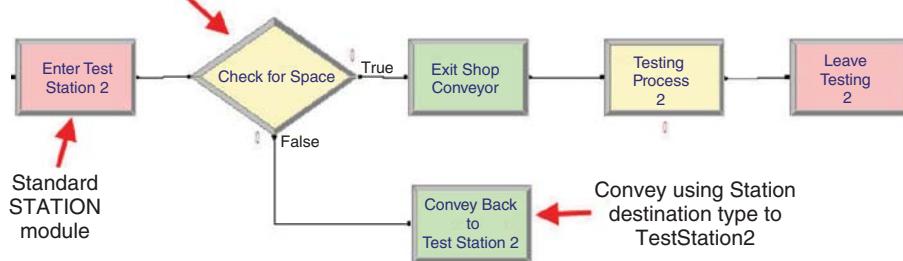
the transfer in option used to exit the conveyor. Then the entity is shunted down one of the lines based on a DECIDE module. In this model, a variable is used to keep track of the last line that an entity was sent to so that the other line will be used in the DECIDE module for the next entity. This implements a simple alternating logic. After being shunted down one of the conveyors, the entity uses the standard ACCESS, CONVEY, STATION, EXIT logic before being disposed.

With a little imagination, a more sophisticated sorting station can be implemented. For example, number of conveyors that handle the parts by destination might be used in the system. When the parts are created, they are given an attribute indicating their destination station. When they get to the sorting station, a DECIDE module can be used to pick the correct conveyor handling their destination.

**9.4.3.3 Processing While on the Conveyor** In the bicycle example from *Smarts101.doe*, the workers can be conceptualized as moving along with the conveyor as they assemble the bicycles. In many systems, it is common for a machine to be positioned to use the surface of the conveyor as its work area. For example, in circuit board manufacturing, the circuit boards ride on a conveyor through chip placement machines. While in the machine, the conveyor stops to do the insertion. These types of situations can be easily modeled in Arena™ by not exiting the conveyor while undergoing a PROCESS module.

Arena's SMART file *Smarts103.doe* contains an example of this type of modeling. In this example, as shown in Figure 9.42, packages are created for the packing area, where they access and convey on a conveyor to the pack and label station. The conveyor consists of two segments attached at the pack and label station. Once at the Labeling Area station, the entity enters a PROCESS module, which requires a delay and the labeling machine. Notice that the entity did not exit the conveyor. The "Loading Area" module is a simple STATION module. After completing the processing, the entity proceeds to be conveyed to the shipping area, where it exits the conveyor. In this example, the conveyor is a nonaccumulating conveyor. If you run the model, you will see that the whole conveyor stops when the entity is being processed. If an accumulating conveyor is used, the packages would queue up behind the labeling machine.

IF NQ(Testing Process 2.Queue) == 0, then proceed into the station and exit the conveyor, else convey back around without exiting the conveyor.



**Figure 9.43** Test and repair system with recirculation conveyor.

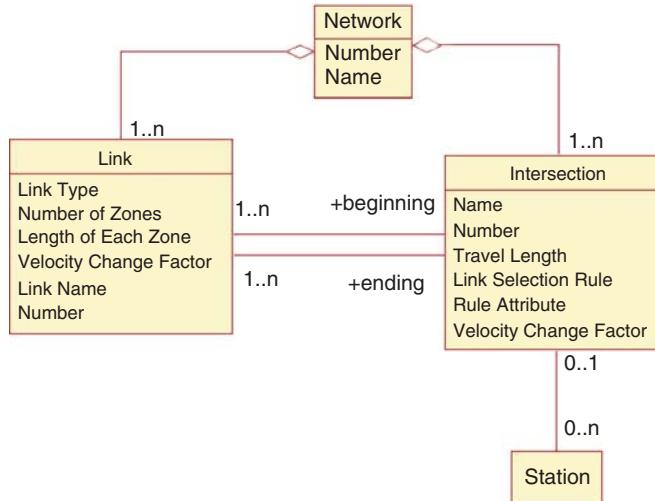
**9.4.3.4 Recirculating Conveyors** A recirculating conveyor is a loop conveyor in which the entities may ride on the conveyor until there is adequate space at their desired location. In essence, the conveyor is used as space to hold the entities that cannot get off due to inadequate space at the necessary station. For example, suppose that in the test and repair shop, there was only space for one waiting part at test station 2. Thus, the size of queue in test station 2 can be at most 1.

The previous model can be easily modified to handle this situation by not using an ENTER module at test station 2. Figure 9.43 shows the changes to the logic for test station 2 to handle this situation. As can be seen in the figure, the ENTER module has been replaced with a combination of STATION, DECIDE, EXIT, and CONVEY. The DECIDE module checks the queue at the Testing Process 2 module. If a part is not in the queue, then the arriving part can exit the station and try to seize the testing resources. If there is a part waiting, then the arriving part is sent to the CONVEY module to be conveyed back around to test station 2. As long as there is a path back to the station, this is perfectly fine (as in this case). The modified model is available with the files for this chapter in the file called *Repair-ShopWithRecirculatingConveyorsWithAnimation.doe*. If you run the model and watch the animation, you will see much more parts on the conveyor because of the recirculation.

There are still a number of issues related to conveyor modeling that have not discussed, especially the use of the specialized variables defined for conveyors. You should refer to the Arena's Variables Guide or to Arena's help system under conveyor variables for more information on this topic. Arena™ also has a number of other SMART files that illustrate the use of conveyors. For example, you might want to explore SMART file *Smarts105.doe* to better understand entity size and how it relates to cells on the conveyor. Conveyors allow the modeling of space through the use of cells. The next section examines how Arena™ models space when it is relevant for transporter modeling.

## 9.5 MODELING GUIDED PATH TRANSPORTERS

This section presents Arena's modeling constructs for automated guided vehicle (AGV) systems. An AGV is an autonomous battery-powered vehicle that can be programmed to move between locations along paths. A complete discussion of AGV systems is beyond the scope of this text. The interested reader should refer to standard texts on material-handling or manufacturing systems design for a more complete introduction to the technology. For example, the texts by Askin and Standridge [1993] and Singh [1996] have complete chapters dedicated to the modeling of AGV systems. For the purposes of this section, the discussion will



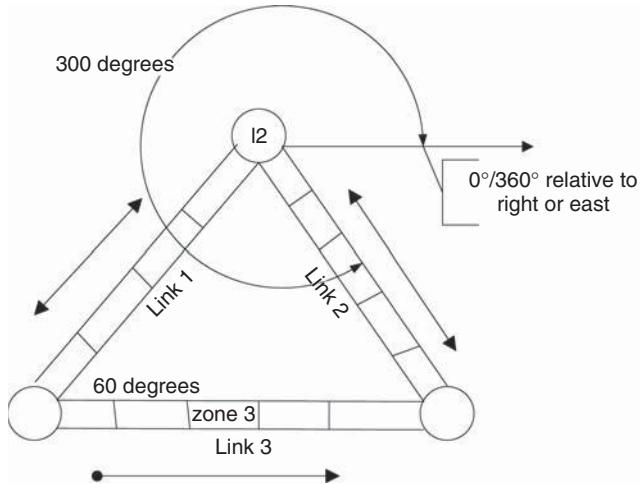
**Figure 9.44** Relational diagram for guided vehicle networks.

be limited to AGV systems that follow a prescribed path (e.g., tape and embedded wires) to make things simpler. There are newer AGVs that do not rely on following a path, but rather are capable of navigating via sensors within buildings, see, for example, Rossetti and Seldanari [2001].

When modeling with guided transporters, the most important issue to understand is that the transporters can now compete with each other for the space along their paths. The space along the path must be explicitly modeled (like it was for conveyors) and the size of the transporter matters. Conceptually, a guided transporter is a mobile resource that moves along a fixed path and competes with other mobile resources for space along the path. When using a guided transporter, the travel path is divided into a network of links and intersections. This is called the vehicle's path network. Figure 9.44 illustrates the major concepts in guided vehicle modeling: networks, links, intersections, and stations.

A link is a connection between two points in a network. Every link has a beginning intersection and an ending intersection. The link represents the space between two intersections. The space on the link is divided into an integer number of zones. Each zone has a given length as specified in a consistent unit of measure. All zones have the same length. The link can also be of a certain type (bidirectional, spur, unidirectional). A bidirectional link implies that the transporter is allowed to traverse in both directions along the link. A spur is used to model “dead ends,” and unidirectional links restrict traffic to one direction. A velocity change factor can also be associated with a link to indicate a change in the basic velocity of the vehicle while traversing the link. For example, for a link going through an area with many people present, you may want the velocity of the transporter to automatically be reduced. Figure 9.45 illustrates a simple three-link network.

In the figure, Link 1 is bidirectional with a beginning intersection labeled I<sub>1</sub> and an ending intersection labeled I<sub>2</sub>. Link 1 consists of four zones. There can also be a direction of travel specified. The beginning direction and the ending direction of the link can be used to define the direction of the link (in degrees) as it leaves the beginning intersection and as it enters the ending intersection. The direction entering the ending intersection defaults to the direction leaving the link's beginning intersection. Thus, the beginning direction for Link 1



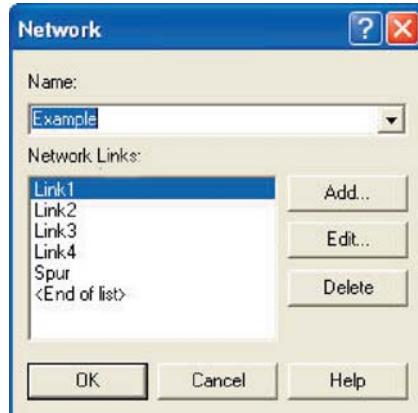
**Figure 9.45** Example of three-link network.

in the figure is  $60^\circ$ . Both 0 and 360 represent the east or right. The beginning direction for Link 2 is  $300^\circ$ . Think of it this way, the vehicle has to turn to go onto Link 2. It would have to turn  $60$  to get back to zero and then another  $60$  to head down Link 2. Since the degrees are specified from the right, this means that the link 2 is  $300^\circ$  relative to an axis going horizontally through intersection I2. The direction of travel is only relevant if the acceleration or deceleration of the vehicles as they make turns is important to the modeling.

In general, an intersection is simply a point in the network; however, the representation of intersections can be more detailed. An intersection also models space: the space between two or more links. Thus, as indicated in Figure 9.44, an intersection may have a length, a velocity change factor, and link selection rule. For more information on the detailed modeling of intersections, you should refer to the INTERSECTIONS element from Arena's Elements template. Also, the books by [Banks et al. (1995)] and [Pegden et al. (1995)] cover the SIMAN blocks and elements that underlie guided path transporter modeling within Arena™. Clearly, a fixed path network consists of a set of links and the associated intersections. Also from Figure 9.44, a station may be associated with an intersection within the network and an intersection may be associated with many stations. Transporters within a guided path network can be sent to specific stations, intersections, or zones. Only the use of stations will be illustrated here.

The new Arena™ constructs that are to be discussed include the NETWORK and NETWORK LINK modules on the Advanced Transfer template. The NETWORK module allows the user to specify a set of links for a path network. Figure 9.46 shows the NETWORK module. To add links, the user presses the add button or uses the spreadsheet dialog box entry. The links added to the NETWORK module either must already exist or must be edited via the NETWORK LINK module. The NETWORK LINK module permits the definition of the intersections on the links, the directional characteristics of the links, number of zones, zone length, and velocity change factor. Figure 9.47 illustrates the NETWORK LINK module. Notice that the spreadsheet view of the module makes editing multiple links relatively easy.

Now, let us take a look at a simple example. In this example, parts arrive to an entry station every 25 minutes, where they wait for one of two available AGVs. Once loaded onto



**Figure 9.46** NETWORK module for guided path modeling.

	Name	Type	Beginning Intersection	Ending Intersection	Beginning Direction	Ending Direction	Number Of Zones	Zone Length	Velocity Change Factor
1	Link1	Unidirectional	I1	I2	0		4	12	1.0
2	Link2	Unidirectional	I2	I3	270		5	12	1.0
3	Link3	Unidirectional	I3	I4	180		4	12	1.0
4	Link4	Unidirectional	I4	I1	90		6	12	1.0
5	Spur	Spur	I4	I5	270		1	36	1.0
6	Link5	Spur	I2	I6	0		1	6	1.0
7	Link6	Spur	I3	I7	0		1	6	1.0

**Figure 9.47** NETWORK LINK module.

the AGV (takes 20 minutes), they are transported to the exit station. At the exit station, the item is unloaded, again this takes 20 minutes. The basic layout of the system is shown in Figure 9.48. There are seven intersections and seven links in this example. The distances between the intersections are shown in the figure.

In modeling this situation, you need to make a number of decisions regarding transporter characteristics, the division of space along the links, the directions of the links, and the zone control of the links. Let us assume that the item being transported is rather large and heavy. That is why it takes so long to load and unload. Because of this, the transporter is a cart that is 6 feet in length and only travels 10 feet per minute. There are two carts available for transport. Figure 9.49 shows the TRANSPORTER module for using guided path transporters. When the “Guided” type is selected, additional options become available.

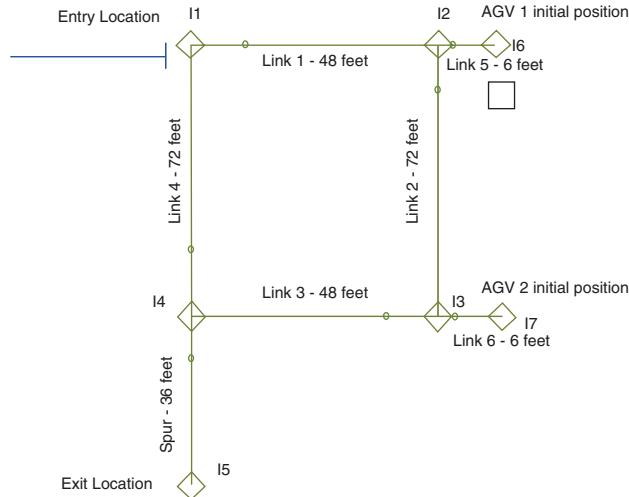


Figure 9.48 Layout for simple AGV example.

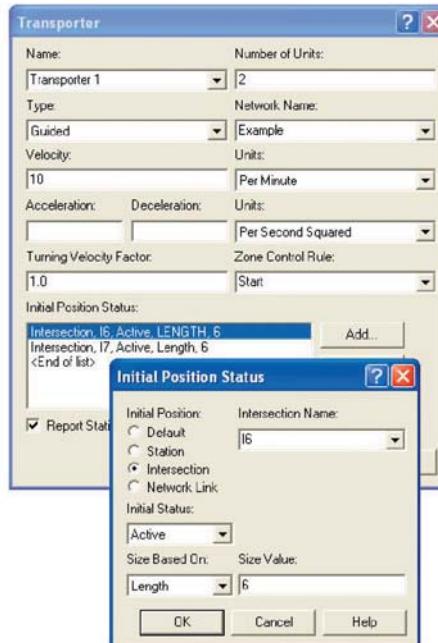


Figure 9.49 TRANSPORTER module for simple AGV example.

Because transporters take space on the path, the choice of initial position can be important. A transporter can be placed at a station that has been associated with an intersection, at an intersection, or on a particular zone on a link. The default option will place the transporter at the first network link specified in the NETWORK module associated with the transporter. As long as you place the transporters so that deadlock (to be discussed shortly) does not happen, everything will be fine. In many real systems, the initial location of the

vehicles will be obvious. For example, since AGVs are battery powered, they typically have a charging station. Because of this, most systems will be designed with a set of spurs to act as a “home base” for the vehicles. For this simple system, intersections I6 and I7 will be used as the initial position of the transporters. All that is left is to indicate that the transporter is active and specify a length for the transporter of 6 feet. To be consistent, the velocity is specified as 10 feet per minute. Before discussing the “Zone Control Rule,” how to specify the zones for the links needs to be discussed.

Guided path transporters move from zone to zone on the links. The size of the zone, the zone control, and the size of the transporter govern how close the transporters can get to each other while moving. Zones are similar to the cell concept for conveyors. The selection of the size of the zones is really governed by the physical characteristics of the actual system. In this example, assume that the carts should not get too close to each other during movement (perhaps because the big part overhangs the cart). In this example, let us choose a zone size of 12 feet. Thus, while the cart is in a zone, it essentially takes up half of the zone. If you think of the cart as sitting at the midpoint of the zone, then the closest the carts can be is 6 feet (3 feet to the front and 3 feet to the rear).

In Figure 9.47, Link 1 consists of four zones of 12 feet in length. Remember the length is whatever unit of measure you want, as long as you are consistent in your specification. With these values, the maximum number of vehicles that could be on Link 1 is four. The rest of the zone specifications are also given in Figure 9.47. Now, you need to decide on the direction of travel permitted on the links. In this system, let us assume that the AGVs will travel clockwise around the figure. Figure 9.47 indicates that the direction of travel from I1 along Link 1 is  $0^\circ$  (to the east). Then, Link 2 has a direction of  $270^\circ$ , Link 3 has a direction of  $180^\circ$ , and Link 4 has a direction of  $90^\circ$ . The spur link to the exit station has a direction of  $270^\circ$  (to the south). Thus, a clockwise turning of the vehicle around the network is achieved. In general, unless you specify the acceleration/deceleration and turning factor for the vehicle, the specification of these directions does not really matter. It has been illustrated here just to indicate what the modeling entails. When working with a real AGV system, these factors can be discerned from the specification of the vehicle and the physical requirements of the system.

To understand why the direction of travel and spurs are important, you need to understand how the transporters move from zone to zone and the concept of deadlock. Zone control governs how the transporter moves from zone to zone. There are three types of control provided by Arena<sup>TM</sup>: Start (of next zone), End (of next zone), or distance units  $k$  (into next zone). The START rule specifies that the transporter will release its backward-most zone when it starts into its next zone. This allows following vehicles to immediately begin to move into the released zone. Figure 9.50 illustrates this concept. In this case, the vehicle is contained in only one zone. In general, a vehicle could cover many zones. The END rule specifies that the transporter will release its backward-most zone after it has moved into the next zone (or reached the end of the next zone). This prevents a following vehicle from moving into the transporter’s current zone until it is fully out of the zone. This decision is not critical to most system operations, especially if the vehicle size is specified in terms of zones. The release at end form of control allows for more separation between the vehicles when they are moving. In the distance units  $k$  (into next zone) rule, the transporter releases its backward-most zone after traveling the  $k$  distance units through the next zone. Since, in general, intersections can be modeled with a traversal distance. These rules also apply to how the vehicles move through intersections and onto links. For more details about the

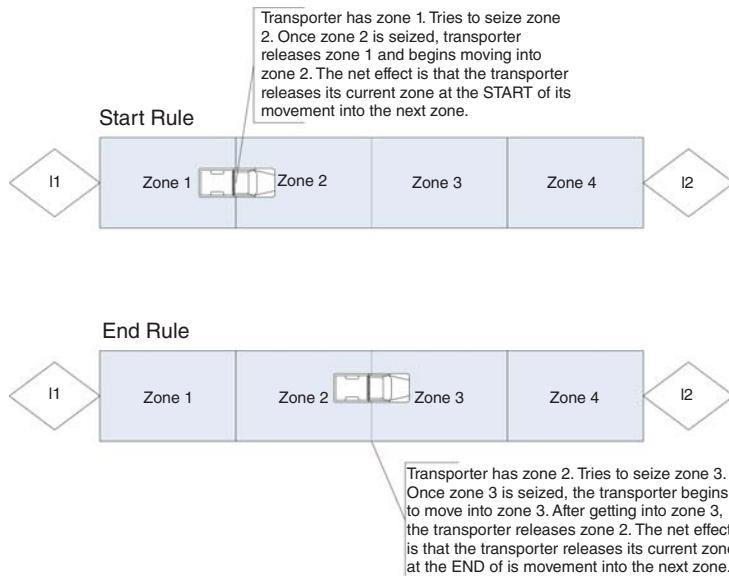


Figure 9.50 Illustrating zone control.

operation of these rules, you should refer to Pegden et al. [1995] for a discussion of the underlying SIMAN constructs.

As indicated in the zone control discussion, guide path transporters move by seizing and releasing zones. Now consider the possibility of a bidirectional link with two transporters moving in opposite directions. What will happen when the transporters meet? They will crash! No, not really, but they will cause the system to get in a state of deadlock. Deadlock occurs when one vehicle requires a zone currently under the control of a second vehicle, and the second vehicle requires the zone held by the first vehicle. In some cases, Arena™ can detect when this occurs during the simulation run. If it is detected, then Arena™ will stop with a run time error. In some cases, Arena™ cannot detect deadlock. For example, suppose a station was attached to I4 and a transporter (T1) was sent directly to that station for some processing. Also, suppose that the part stays on transporter (T1) during the processing so that transporter (T1) is not released at the station. Now, suppose that transporter (T2) has just dropped off something at the exit point, I5, and that a new part has just come in at I1. Transporter (T2) will want to go through I4 on its way to I1 to pick up the new part. Transporter (T2) will wait until transporter (T1) finishes its processing at I4 and if transporter (T1) *eventually* moves, there will be no problem. However, suppose transporter (T1) is released at I4 and no other parts arrive for processing. Transporter (T1) will become idle at the station attached to I4 and will continue to occupy the space associated with I4. Since, no arriving parts will ever cause transporter (T1) to move, transporter (T2) will be stuck. There will be the one part waiting for transporter (T2), but transporter (T2) will never get there to pick it up.

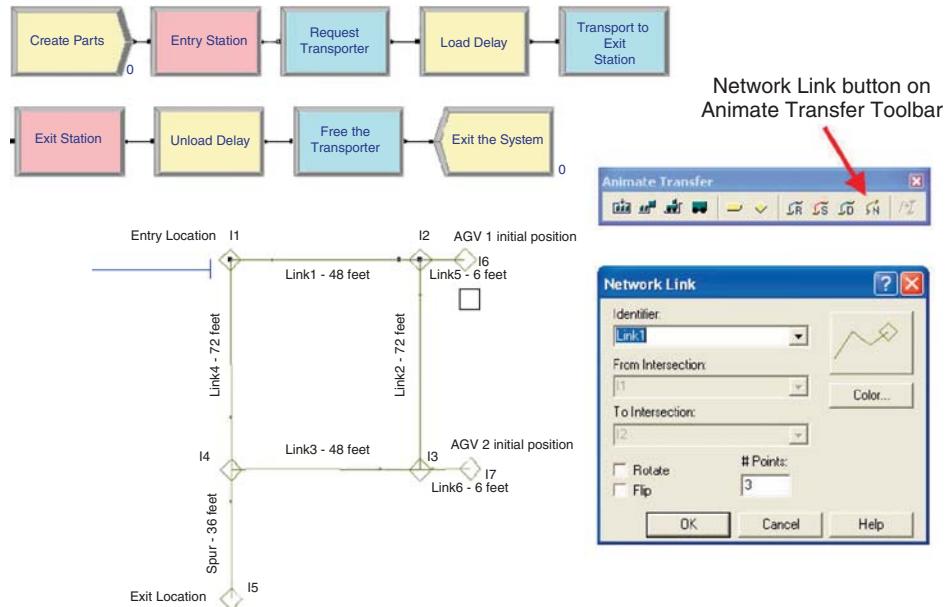
Obviously, this situation has been concocted to illustrate a possible undetected deadlock situation. The point is: care must be taken when designing guided path networks so that deadlock is prevented by design or that special control logic is added to detect and react to various deadlock situations. For novice modelers, deadlock can seem troublesome,

especially if you do not think about where to place the vehicles and if you have bidirectional links. In most real systems, you should not have to worry about deadlock because you should design the system not to have it in the first place! In the simple example, all unidirectional links were used along with spurs. This prevents many possible deadlock situations. To illustrate what is meant by design, consider the concocted undetectable deadlock situation. A simple solution to prevent it would be to provide a spur for transporter (T1) to receive the processing that was occurring at the station associated with I4 and instead associate the processing station with the intersection at the end of the new spur. Thus, during the processing, the vehicle will be out of the way. If you were designing a real system, wouldn't you do this common sense thing anyway? Another simple solution would be to always send idle vehicles to a staging area or home base that gets them out of the way of the main path network.

To finish up this discussion, let us briefly discuss the operation of spurs and bidirectional links. This discussion is based in part on [Pegden et al. (1995, p 394)] Notice that the link from I4 to I5 is a spur in the previous example. A spur is a special type of link designed to help to prevent some deadlock situations. A spur models a “dead end” and is bidirectional. If a vehicle is sent to the intersection at the end of the spur (or to a station attached to the end intersection), then the vehicle will need to maintain control of the intersection associated with the entrance to the spur, so that it can get back out. If the vehicle is too long to fit on the entire spur, then the vehicle will naturally control the entering intersection. It will block any vehicles from moving through the intersection. If the spur link is long enough to accommodate the size of the vehicle, the entering intersection will be released when the vehicle moves down the spur; however, any other vehicles that are sent to the end of the spur will not be permitted to gain access to the entering intersection until all zones in the spur link are available. This logic only applies to vehicles that are sent to the end of the spur. Any vehicles sent to or through the entering intersection of the spur are still permitted to gain access to the spur’s entering intersection. This allows traffic to continue on the main path while vehicles are down the spur. Unfortunately, if a vehicle stops at the entering intersection and then never moves, the situation described in the concocted example occurs.

Bidirectional links work similarly to unidirectional links, but a vehicle moving on a bidirectional link must have control of both the entering zone on the link and the direction of travel. If another vehicle tries to move on the bidirectional link, it must be moving in the same direction as any vehicles currently on the link; otherwise, it will wait. Unfortunately, if a vehicle (wanting to go down a bidirectional link) gains control of the intersection that a vehicle traveling on the bidirectional link needs to exit the link, deadlock may be possible. Because it is so easy to get in deadlock situations with bidirectional links, you should think carefully about their use when using them within your models.

The animation of guided path transporters is very similar to regular transporters. On the Animation Transfer toolbar, the Network Link button is used to lay down intersections and the links between the intersections as shown in Figure 9.51. Arena™ will automatically recognize and fill the pull down text-boxes with the intersections and links that have been defined in the NETWORK LINK module. Although the diagram is not to scale, the rule or scale button comes in very handy when laying out the network links to try to get a good mapping to the physical distances in the problem. Once the network links have been placed, the animation works just like you are used to for transporters. The file *SimpleAGVExample.doe* is available for this problem. In the model, the parts are created and sent to the entry station, where they request a transporter, delay for the loading, and then transport to the exit station. At the exit station, there is a delay for the unloading before the transporter



**Figure 9.51** Animating guided path transporters.

is freed. The REQUEST, TRANSPORT, and FREE modules all work just like for free path transporters.

You should try running the model. You will see that the two transporters start at the specified intersections and you will clearly see the operation of the spur to the exit point as it prevents deadlock. The second transporter will wait for the transporter already on the spur to exit the spur before heading down to the exit point. One useful technique is to use the View > Layers menu to view the transfer layers during the animation. This will allow the intersections and links to remain visible during the model run.

This section has provided a basic introduction to using guided path transporters in Arena™ models. There are many more issues related to the use of these constructs that have not been covered:

1. The construction and use of the shortest path matrix for the guided path network. Since there may be multiple paths through the network, Arena™ uses standard algorithms to find the shortest path between all intersections in the network. This is used when dispatching the vehicles.
2. Specialized variables are used in guided path networks. These variables assist with indicating where the vehicle is within the network, accessing information about the network within the simulation, (e.g., distances and beginning intersection for a link), status of links, intersections, etc. Arena's help system under "Transporter Variables" discusses these variables and a discussion of their use can also be found in Pegden et al. [1995].
3. Changing of vehicle speeds, handling failures, changing vehicle size, redirecting vehicles through portions of the network, and link selection rules.

More details on these topics can be found in Banks et al. [1995] and Pegden et al. [1995]. Advanced examples are also given in both of those texts; Pegden et al. [1995] provided the possible use of guided path constructs for modeling automated storage and retrieval systems (AS/RS) and overhead cranes.

## 9.6 SUMMARY

This chapter provided an introduction to the modeling of entity transfer. In particular, the use of resource-constrained transfer, transporters, and conveyors from an Arena™ modeling perspective were all discussed. Many of these concepts are especially useful in modeling manufacturing and distribution systems. The animation of entity transfer was also discussed through a number of embellishments to the test and repair model. While still somewhat crude, the animation concepts presented in this chapter are very effective for validating and selling an Arena™ model to decision makers. With more time, effort, patience, and artistic skill, you can easily create compelling animations for your simulations.

A large portion of the modeling concepts within Arena™ have now been covered. There are a number of miscellaneous (yet important) concepts that still need to be discussed. In particular, Chapter 10 will discuss more advanced modeling of resources (including staffing and breakdowns) and the basics behind Arena's cost modeling constructs.

## EXERCISES

- 9.1 Reconsider Exercise 8.16. Assume now that the travel time between the two systems is more significant. The travel time is distributed according to a triangular distribution with parameters (1, 2, 4) minutes.
  - (a) Model the system assuming that the worker from the first station moves the parts to the second station. The movement of the part should be given priority if there is another part waiting to be processed at the first station.
  - (b) Model the system with a new worker (resource) to move the parts between the stations.
  - (c) From your model, estimate the total system time for the parts, the utilization of the workers, and the average number of parts waiting for the workers. Run the simulation for exactly 20,000 minutes with a warm-up period of 5000 minutes.
- 9.2 Reconsider part (b) of Exercise 9.1. Instead of immediately moving the part, the transport worker waits until a batch of five parts has been produced at the first station. When this occurs, the worker moves the batch of parts to the second station. The parts are still processed individually at the second station. From your model, estimate the total system time for the parts, the utilization of the workers, and the average number of parts waiting for the workers. Run the simulation for exactly 20,000 minutes with a warm-up period of 5000 minutes.
- 9.3 Redo Exercise 8.19 using resource-constrained transfer. Assume that there are two workers at the diagnostic station, one worker per testing station, and three workers at the repair station. Thus, there are a total of eight workers in the system. Furthermore, assume that any of these eight workers are capable of moving parts between the stations. For example, when a part completes its operation at the diagnostic station,

any worker in the system can carry the part to the next station. When a part requires movement, it will wait for the next available idle worker to complete the movement. Assume that parts waiting for processing at a station will be given priority over parts that require movement between stations. Build a simulation model that can assist the company in assessing the risks associated with the new contract under this resource-constrained situation.

- 9.4 Redo Exercise 8.20 assuming that there is a pool of three workers who perform the transport between the stations. Assume that the transport time is triangularly distributed with parameters (2, 4, 6) all in minutes. Make an assessment for the company for the appropriate number of workers to have in the transport worker pool.
- 9.5 In Section 9, the test and repair system was analyzed with three transporters. Reanalyze this situation and recommend an appropriate number of transport workers.
- 9.6 Reconsider part (b) of Exercise 9.1. Instead of modeling the problem with a resource, the problem should be modeled with a transporter. Assume that there is one transporter (fork truck) that must move the parts between the two stations. The distance between the two stations is 100 meters and the fork truck's velocity between the stations is triangularly distributed with parameters (25, 50, 60) in meters per minute.
  - (a) Model the system and estimate the total system time for the parts, the utilization of the workers, and the average number of parts waiting for the fork truck. Run the simulation for exactly 20,000 minutes with a warm-up period of 5000 minutes.
  - (b) Instead of immediately moving the part, the fork truck waits until a batch of five parts has been produced at the first station. When this occurs, the fork truck is called to move the batch of parts to the second station. The parts are still processed individually at the second station. Redo the analysis of part (a) for this situation.
- 9.7 Reconsider Exercise 9.4 using transporters. The distances between the four stations (in feet) are given in the following table. After the parts finish the processing at the last station of their sequence, they are transported to an exit station, where they begin their shipping process.

Station	Destination				
	A	B	C	D	Exit
Origin	A	–	50	60	80
	B	55	–	25	55
	C	60	20	–	70
	D	80	60	65	–
Exit	100	80	35	40	–

There are two fork trucks in the system. The velocity of the fork trucks varies within the facility due to various random congestion effects. Assume that the fork truck's velocity between the stations is triangularly distributed with parameters (3, 4, 5) miles per hour. Simulate the system for 30,000 minutes and discuss the potential bottlenecks in the system.

- 9.8 Reconsider the test and repair example from Section 9. In this problem, the workers have a home base that they return to whenever there are no more waiting requests rather than idling at their last drop-off point. The home base is located at the center of the shop. The distances from each station to the home base are given in as follows.

		Destination					
	Station	Diagnostics	Test 1	Test 2	Test 3	Repair	Home base
Origin	Diagnostics	–	40	70	90	100	20
	Test 1	43	–	10	60	80	20
	Test 2	70	15	–	65	20	15
	Test 3	90	80	60	–	25	15
	Repair	110	85	25	30	–	25
	Home base	20	20	15	15	25	–

Update the animation so that the total number of waiting requests for movement is animated by making it visible at the home base of the workers. Rerun the model using the same run parameters as the chapter example and report the average number of requests waiting for transport. Give an assessment of the risk associated with meeting the contract specifications based on this new design.

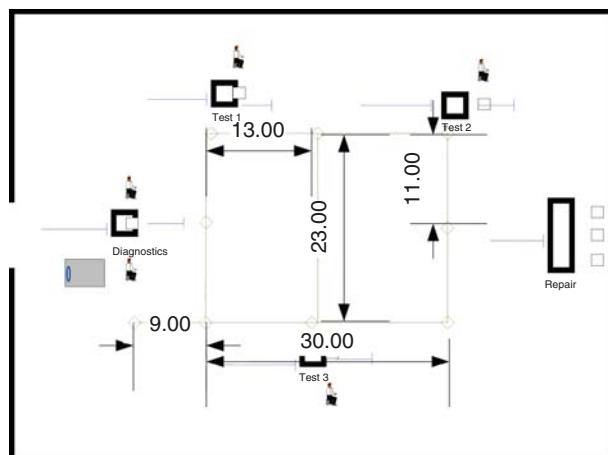
- 9.9 Reconsider Exercise 8.35 with transporters to move the parts between the stations. The stations are arranged sequentially in a flow line with 25 meters between each station, with the distances provided as follows:

		Destination			
	Station	1	2	3	Exit
Origin	1	–	25	50	75
	2	25	–	25	50
	3	50	25	–	25
	Exit	75	50	25	–

Assume that there are two transporters available to move the parts between the stations. Whenever a part is required at a downstream station and a part is available for transport, the transporter is requested. If no part is available, no request is made. If a part becomes available and a part is needed, then the transporter is requested. The velocity of the transporter is considered to be  $\text{TRIA}(22.86, 45.72, 52.5)$  in meters per minute. By using the run parameters of Exercise 8.34, estimate the effect of the transporters on the throughput of the system.

- 9.10 Three independent conveyors deliver 1 foot parts to a warehouse. Once inside the warehouse, the conveyors merge onto one main conveyor to take the parts to shipping. Parts arriving on conveyor 1 follow a Poisson process with a rate of 6 parts per minute. Parts arriving on conveyor 2 follow a Poisson process with a rate of 10 parts per minute. Finally, parts arriving on conveyor 3 have an interarrival time uniformly distributed between 0.1 and 0.2 minutes. Conveyors 1 and 2 are accumulating conveyors and are 15 and 20 feet long, respectively. They both run at a velocity of 12 feet per minute. Conveyor 3 is an accumulating conveyor and is 10 feet long. It runs at a velocity of 20 feet per minute. The main conveyor is 25 feet long and is an accumulating conveyor operating at a speed of 25 feet per minute. Consider the parts as being disposed after they reach the end of the main conveyor. Simulate this system for 480 minutes and estimate the accessing times and the average number of parts waiting for the conveyors.

- 9.11 This problem is based on problem E10.6 of Banks *et. al.* [1995]. Used with permission. A 140-feet, nonaccumulating loop conveyor is used to feed jobs to seven workers who are evenly spaced around a loop. Jobs are loaded onto the conveyor 5 feet prior to worker #1. An incomplete job will continue to move around the conveyor until an idle worker is encountered. Once encountered, the job is unloaded from the conveyor, processed, and the completed job is reloaded on the conveyor. Completed jobs are unloaded from the conveyor 5 feet after worker #7. Jobs arrive exponentially with a mean time between arrivals of 2.2 minutes. Job processing is lognormally distributed with a mean of 14 minutes and a variance of 3.6 minutes squared for all workers. The loop conveyor has a velocity of 15 feet per minute. Arriving jobs wait in a storage buffer until space on the loop conveyor becomes available.
- Simulate for 1 day of continuous operation and determine the utilization of the conveyor, the average number of jobs on the conveyor, the average time spent in the system, and the utilization of each worker.
  - Suppose that the conveyor is subject to breakdowns. The time between breakdowns is exponentially distributed with a mean time of 9 hours. The time to repair is also exponentially distributed with a mean of 7 minutes. Simulate for 1 day of continuous operation and determine the utilization of the conveyor, the average number of jobs on the conveyor, the average time spent in the system, and the utilization of each worker. Hint: Consider the use of the STOP and START modules.
- 9.12 Reconsider Exercise 8.35 with conveyors to move the parts between the stations. Suppose that a single nonaccumulating conveyor of length 75 meters, with 25-meter segments between each of the stations is used in the system. When a part is needed by a downstream station, it is loaded onto the conveyor if available. The load time takes between 15 and 45 seconds, uniformly distributed. The speed of the conveyor is 5 meters per minute. If a part is produced and the downstream station requires the part, it is loaded onto the conveyor. By using the run parameters of Exercise 8.35, estimate the effect of the transporters on the throughput of the system.
- 9.13 This problem considers the use of AGVs for the test and repair system of this chapter. The layout of the proposed system is given below with all measurements in meters.



There is only one AGV in this system. It is 1 meter in length and moves at a velocity of 30 meters per minute. Its home base is at the dead end of the 9 meter spur. Since there will be only one AGV, all links are bidirectional. The design associates the stations with the nearest intersection on the path network.

- (a) Simulate the system for 10 replications of 4160 hours. Estimate the chance that the contract specifications are met and the average system time of the jobs. In addition, assess the queuing for and the utilization of the AGV.
- (b) Consider the possibility of having a two-AGV system. What changes to your design do you recommend? Be specific enough that you could simulate your design.

- 9.14 *This problem is based on an example on page 223 of Pegden et. al. [1995]. Used with permission.* Reconsider Exercise 8.18 with the use of transporters. Assume that all parts are transferred by using two fork trucks that travel at an average speed of 150 feet per minute. The distances (in feet) between the stations are provided in the table below.

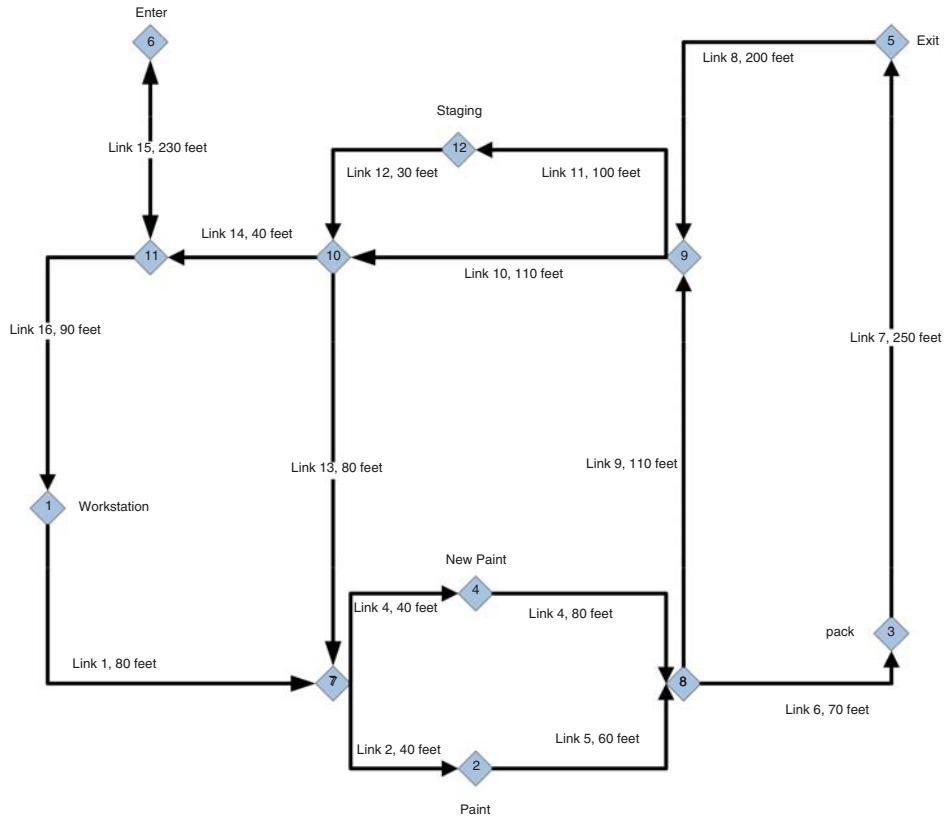
	Enter	Workstation	Paint	New Paint	Pack	Exit
Enter	—	325	445	445	565	815
Workstation		—	120	130	240	490
Paint			—	250	120	370
New Paint				—	130	380
Pack					—	250
Exit						—

The distances are symmetric. Both the drop-off and pickup points at a station are at the same physical location. Once the truck reaches the pickup/drop-off station, it requires a load/unload time of 2 minutes.

Analyze this system to determine any potential bottleneck operations. Report on the average flow times of the parts as a whole and individually. Also obtain statistics representing the average number of parts waiting for allocation of a transporter and the number of busy transporters. Run the model for 600,000 minutes with a 50,000-minute warm-up period. Develop an animation for this system that illustrates the transport and queuing within the system.

- 9.15 *This problem is based on an example on page 381 of Pegden et. al. (1995). Used with permission.* Reconsider Problem 9.14 with the use of AGVs. There are now three AGVs that travel at a speed of 100 feet per minute to transport the parts within the system. To avoid deadlock situations the guided path network has been designed for one way travel. In addition, to prevent an idle vehicle from blocking vehicles performing transports, a staging area (Intersection 12) has been placed on the guided path network. Whenever a vehicle completes a transport, a check should be made of the number of requests pending for the transporters. If there are no requests pending, the AGV should be sent to the staging area. If there are multiple vehicles idle, they should wait on Link 11. Link 15 is a spur from intersection 6 to intersection 11. The stations for the operation on the parts should be assigned to the intersections as shown in the figure. The links should all be divided into zones of 10 feet each. The initial

position of the vehicles should be along Link 11. Each vehicle is 10 feet in length or 1 zone. The release at start form of zone control should be used. The guided path network for the AGVs is given below:



Analyze this system to determine any potential bottleneck operations. Report on the average flow times of the parts as a whole and individually. Also obtain statistics representing the average number of parts waiting for allocation of a transporter and the number of busy transporters. Run the model for 600,000 minutes with a 50,000-minute warm-up period. Develop an animation for this system that illustrates the transport and queuing within the system.

---

# 10

---

## MISCELLANEOUS TOPICS IN ARENA MODELING

### LEARNING OBJECTIVES

- To be able to model non-stationary arrivals using Arena<sup>TM</sup>.
- To be able to model the staffing/scheduling and failure/repair of resources.
- To be able to utilize Arena's entity and resource costing constructs.
- To be able to model with advanced concepts involving picking stations, picking up entities into groups, dropping off entities from groups, and generic station modeling.
- To be able to understand and use some advanced programming concepts.

### 10.1 INTRODUCTION

This chapter tackles a number of miscellaneous topics that can enhance your modeling capabilities. The chapter first examines the modeling of non-stationary arrival processes. This will motivate the exploration of additional concepts in resource modeling, including how to incorporate time-varying resource staffing schedules. This will enable you to better model and tabulate the effects of realistic staff changes within a model. With enhanced resource modeling, you will want to better model the costs associated with entities using the resources within models. Thus, a discussion of how Arena<sup>TM</sup> tabulates entity and resource costs is also presented.

As entities move through the model, they are often formed into groups. Arena's constructs for grouping and ungrouping entities will also be presented. Finally, an introduction to some of the programming aspects of Arena<sup>TM</sup> will be explored (e.g., database,

spreadsheet interaction, Visual Basic for Applications, etc.). Let us get started by looking at how time-dependent arrivals can be generated.

## 10.2 NON-STATIONARY PROCESSES

If a process does not depend on time, it is said to be stationary. When a process depends on time, it is said to be non-stationary. The more formal definitions of stationary/non-stationary are avoided in the discussion that follows.

There are many ways in which you can model non-stationary (time varying) processes in your simulation models. Many of these ways depend entirely on the system being studied and through that study appropriate modeling methods will become apparent. For example, suppose that workers performing a task learn how to more efficiently perform the task every time that they repeat the task. The task time will depend on the time (number of previously performed tasks). For this situation, you might use a learning curve model as a basis for changing the task times as the number of repetitions of the task increases.

As another example, suppose the worker's availability depends on a schedule, such as having a 30-minute break after accumulating so much time. In this situation, the system's resources are dependent on time. Modeling this situation is described later in this chapter.

Let us suppose that the following situation needs to be modeled. Workers are assigned a shift of 8 hours and have a quota to fill. Do you think that it is possible that their service times would, on an average, be less during the latter part of the shift? Sure. They might work faster in order to meet their quota during the latter part of the shift. If you did not suspect this phenomenon and did perform a time study on the workers in the morning and fit a distribution to these data, you would be developing a distribution for the service times during morning. If you applied this distribution to the entire shift, then you may have a problem matching your simulation throughput numbers to the real throughput.

As you can see, if you suspect that a non-stationary process is involved, then it is critical that you also record the time that the observation was made. The time-series plot that was discussed for input modeling helps in assessing non-stationary behavior; however, it only plots the order in which the data were collected. If you collect 25 observations of the service time every morning, you might not observe non-stationary behavior. To observe non-stationary behavior, it is important to collect observations across periods of time.

One way to better plan your observations is to randomize when you will take the observations. Work sampling methods often require this. The day is divided into intervals of time and the intervals randomly selected in which to record an activity. By comparing the behavior of the data across the intervals, you can begin to assess whether time matters in the modeling. This is what was done in Example 6.1. Even if you do not divide time into logical intervals, it is good practice to record the date/time for each of the observations that you make so that the observations can later be placed into logical intervals. As you may now be thinking, modeling a non-stationary process will take a lot of care and more observations than a stationary process.

A very common non-stationary process that you have probably experienced is a non-stationary arrival process. In a non-stationary arrival process, the arrival process to the system will vary by time (e.g., by hour of the day). Because the arrival process is a key input to the system, the system will thus become non-stationary. For example, in the drive-through pharmacy example, the arrival of customers occurred according to a Poisson process with mean rate  $\lambda$  per hour.

As a reminder,  $\lambda$  represents the mean arrival rate or the expected number of customers per unit time. Suppose that the expected arrival rate is 5 per hour. This does not mean that the pharmacy will get five customers every hour, but rather that on an average five customers will arrive every hour. Some hours will get more than five customers and other hours will get less. The implication for the pharmacy is that they should expect about five per hour (every hour of the day) regardless of the time of day. Is this realistic? It could be, but it is more likely that the mean number of arriving customers will vary by time of day. For example, would you expect to get five customers per hour from 4 a.m. to 5 a.m., from 12 noon to 1 p.m., from 4 p.m. to 6 p.m.? Probably not! A system like the pharmacy will have peaks and valleys associated with the arrival of customers. Thus, the mean arrival rate of the customers may vary with the time of day. That is,  $\lambda$  is really a function of time,  $\lambda(t)$ .

To more realistically model the arrival process to the pharmacy, you need to model a non-stationary arrival process. Let us think about how data might be collected in order to model a non-stationary arrival process. First, as in the previous service time example, you need to think about dividing the day into logical periods of time. A pilot study may help in assessing how to divide time, or you might simply pick a convenient time division such as hours. You know that the CREATE module requires a distribution that represents the time between arrivals. Ideally, you should collect the time of every arrival,  $T_i$ , throughout the day. Then, you can take the difference among consecutive arrivals,  $T_i - T_{i-1}$  and fit a distribution to the interarrival times. Looking at the  $T_i$  over time may help to assess whether you need different distributions for different time periods during the day. Suppose that you do collect the observations and find that three different distributions are reasonable models, given the following data:

- Exponential,  $E[X] = 60$ , for midnight to 8 a.m.
- Lognormal,  $E[X] = 12$ ,  $\text{Var}[X] = 4$ , for 8 a.m. to 4 p.m.
- Triangular, min = 10; mode = 16; max = 20, for 4 p.m. to midnight

One simple method for implementing this situation is to CREATE a logical entity that selects the appropriate distribution for the current time. In other words, schedule a logical entity to arrive at midnight so that the time between arrival distribution can be set to exponential, schedule an entity to arrive at 8 a.m. to switch to lognormal, and schedule an entity to arrive at 4 p.m. to switch to triangular. Clearly, this varies the arrival process in a time-varying manner. There is nothing wrong with this modeling approach if it works well for the situation. In taking such an approach, you need a lot of data. Not only do you need to record the time of every arrival, but you also need enough arrivals in order to adequately fit distributions to the time between events for various time periods. This may or may not be feasible in practice.

Often in modeling arrival processes, you cannot readily collect the actual times of the arrivals. If you can, you should try, but sometimes you just cannot. Instead, you are limited to a count of the number of arrivals in intervals of time. For example, a computer system records the number of people arriving every hour but not their individual arrival times. This is not uncommon since it is much easier to store summary counts than it is to store all individual arrival times. Suppose that you had count data as shown in Table 10.1.

Of course, how the data are grouped into intervals will affect how the data can be modeled. Grouping is part of the modeling process. Let us accept these groups as appropriate for this situation. Looking at the intervals, you see that more arrivals tend to occur from 12 to 2 p.m. and from 5 to 8 p.m. As discussed in Chapter 6, we could test for this non-stationary

**TABLE 10.1 Example Arrival Counts**

Interval	Mon	Tue	Wed	Thur	Fri	Sat	Sun
12–6 a.m.	3	2	1	2	4	2	1
6–9 a.m.	6	4	6	7	8	7	4
9 a.m.–12 p.m.	10	6	4	8	10	9	5
12–2 p.m.	24	25	22	19	26	27	20
2–5 p.m.	10	16	14	16	13	16	15
5–8 p.m.	30	36	26	35	33	32	18
8 p.m.–12 a.m.	14	12	8	13	12	15	10

behavior and check if the counts depend on the time of day. It is pretty clear that this is the case for this situation. Perhaps this corresponds to when people who visit the pharmacy can get off of work. Clearly, this is a non-stationary situation. What kinds of models can be used for this type of count data?

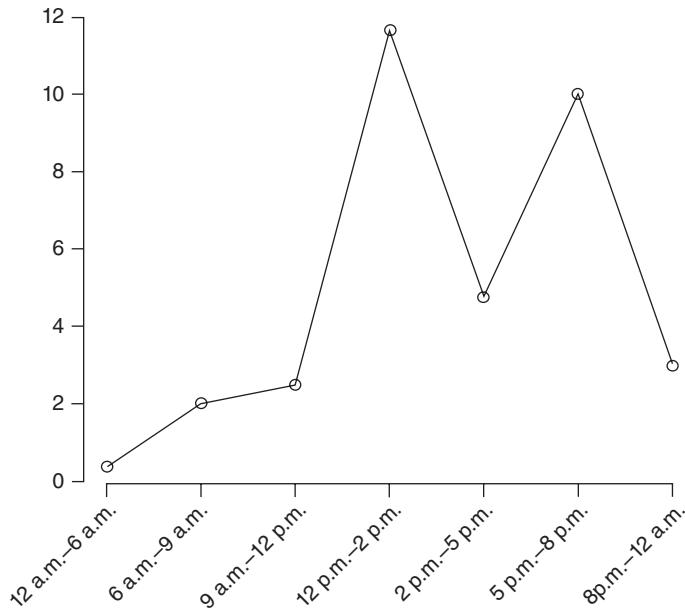
A simple and often reasonable approach to this situation is to check whether the number of arrivals in *each* interval occurs according to a Poisson distribution. If so, then you know that the time between arrivals in an interval is exponential. When you can divide time so that the number of events in the intervals is Poisson (with different mean rates), then you can use a non-stationary or nonhomogeneous Poisson process as the arrival process.

Consider the first interval to be 12 a.m. to 6 a.m.; to check if the counts are Poisson in this interval, there are seven data points (one for every day of the week for the given period). This is not a lot of data to base a chi-square goodness-of-fit test for the Poisson distribution. So, to do a better job at modeling the situation, many more days of data are needed. The next question is: Are all the days the same? It looks like Sunday may be a little different from the rest of the days. In particular, the counts on Sunday appear to be a little less on average than the other days of the week. In this situation, you should also check whether the day of the week matters to the counts. As illustrated in Chapter 6, one method of doing this is to perform a contingency table test.

Let us assume for simplicity that there is not enough evidence to suggest that the days are different and that the count data in each interval is well modeled with a Poisson distribution. In this situation, you can proceed with using a non-stationary Poisson process. From the data you can calculate the average arrival rate for each interval and the rate per hour for each interval as shown in Table 10.2. In the table, 2.14 is the average of the counts across the days for the interval 12 a.m. to 6 p.m. In the rate per hour column, the interval rate has been converted to an hourly basis by dividing the rate for the interval by the number of hours in the interval. Figure 10.1 illustrates the time-varying behavior of the mean arrival rates over the course of a day.

The two major methods by which a non-stationary Poisson process can be generated are thinning and rate inversion.

Both methods will be briefly discussed; however, the interested reader is referred to Law [2006] or Casella and Berger [1997] for more of the theory underlying these methods. Before these methods are discussed, it is important to point out that the naive method of using different Poisson processes for each interval and subsequently different exponential distributions to represent the interarrival times could be used to model this situation by switching the distributions as previously discussed. However, the switching method is not technically correct for generating a non-stationary Poisson process. This is because it will not generate a non-stationary Poisson process with the correct probabilistic underpinnings.

**Figure 10.1** Arrival rate per hour for intervals.**TABLE 10.2** Mean arrival rates for each interval

Interval	Average	Length, hours	Rate per Hour
12 a.m.-6 a.m.	2.14	6	0.36
6 a.m.-9 a.m.	6.0	3	2.0
9 a.m.-12 p.m.	7.43	3	2.48
12 p.m.-2 p.m.	23.29	2	11.645
2 p.m.-5 p.m.	14.29	3	4.76
5 p.m.-8 p.m.	30.0	3	10
8 p.m.-12 a.m.	12.0	4	3.0

Thus, if you are going to model a process with a non-stationary Poisson process, you should use either thinning or rate inversion to be technically correct.

### 10.2.1 Thinning Method

For the thinning method, a stationary Poisson process with a constant rate,  $\lambda^*$ , and arrival times,  $t_i^*$ , is first generated and then potential arrivals,  $t_i^*$ , are rejected with probability:

$$p_r = 1 - \frac{\lambda(t_i^*)}{\lambda^*} \quad (10.1)$$

where  $\lambda(t)$  is the mean arrival rate as a function of time. Often,  $\lambda^*$  is set as the maximum arrival rate over the time horizon, such as:

$$\lambda^* = \max_t \{\lambda(t)\} \quad (10.2)$$

In the example,  $\lambda^*$  would be the mean of the 5 p.m. to 8 p.m. interval,  $\lambda^* = 11.645$ . Exhibit 10.1 is the pseudo-code for implementing the thinning method creates entities at the maximum rate and thins them according to which period is currently active. An array stores each arrival rate for the intervals, and a variable is used to hold the maximum arrival rate over the time horizon. To implement the arrival rate function, the code creates a logical entity that keeps track of the current interval as a variable that can be used to index the array of arrival rates. If the simulation lasts past the time horizon of the arrival rate function, then the variable that keeps track of the interval should be reset to start counting from the first period.

---

### Exhibit 10.1 Pseudo-Code for Thinning Algorithm

---

```

CREATE entities, mean time between arrivals, EXPO(1/\lambda*)
DECIDE with chance, 1 - λ_i^*/λ*
    IF accepted THEN
        send entity to next module
    ELSE
        DISPOSE
    END IF
END DECIDE

CREATE 1 logical entity
ASSIGN
    A: vPeriod = vPeriod + 1
END ASSIGN
DECIDE
    IF vPeriod < vNumPeriods THEN
        DELAY for period length
    ELSE
        vPeriod = 0
    END IF
END DECIDE
GOTO: A

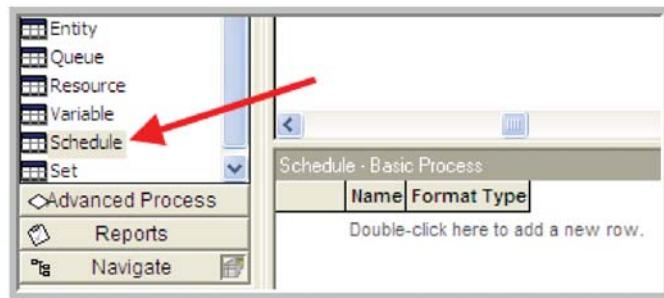
```

---

One of the exercises in this chapter requires implementing the thinning algorithm in Arena™. The theoretical basis for why thinning works can be found in Ross [1997].

#### 10.2.2 Rate Inversion Method

The second method for generating a non-stationary Poisson process is through the rate inversion algorithm. In this method, a  $\lambda = 1$  Poisson process is generated, and the inverse of the mean arrival rate function is used to rescale the times of arrival to the appropriate scale. This section does not discuss the theory behind this algorithm. Instead, see Leemis and Park [2006] for further details of the theory, fitting, and the implementation of this



**Figure 10.2** SCHEDULE data module.

method in simulation. This method is available for piece-wise constant-rate functions in Arena™ by using an Arrivals Schedule associated with a CREATE module. Let us see how to implement the example using an Arrivals Schedule.

The following is available in the Arena™ file *PharmacyModelNSPP.doe* that accompanies this chapter. The first step in implementing a non-stationary Poisson process in Arena™ is to define the intervals and compute the rates per hour as per Table 10.2. It is extremely important to have the rates on a per hour basis. The SCHEDULE module is a data module on the Basic Process panel.

Figure 10.2 illustrates where to find the SCHEDULE in the Basic Process panel. To use the SCHEDULE module, double-click on the data module area to add a new schedule. Schedules can be of type Capacity, Arrival, and Other. Right-clicking on the row and using Edit via Dialog will give you the SCHEDULE dialog window. The following are the text boxes for the SCHEDULE dialog.

**Format Type** Schedules can be defined as a collection of value, duration pairs (Duration) or they can be defined using the calendar editor. The calendar editor allows for the input of a complex schedule that can be best described by a Gregorian calendar (i.e., days and months).

**Type** Capacity refers to time-varying schedules for resources. Arrival refers to non-stationary Poisson arrivals. Other can be used to represent other types of time-delayed schedules.

**Time Units** This field represents how the time units for the durations will be interpreted. The field only refers to the durations, not the arrival rates.

**Scale Factor** This field can be used to easily change all the values in the duration value pairs.

**Durations** The length of time of the interval for the associated value. Once all the durations have been executed, the schedule will repeat unless the last duration is left infinite.

**Value** This field represents either the capacity to be used for the resource or the arrival rate (in entities per hour), depending on which type of schedule was specified.

**Name** Name of the module.

The SCHEDULE module offers a number of ways in which to enter the information required for the schedule. The most straightforward method is to enter each value and duration pair using the Add button on the dialog box. You will be prompted for each pair. In this case, the arrival rate (per hour) and the duration that the arrival rate will last should be entered. The completed dialog entries are shown in Figure 10.3.

Note that the arrival rates are given per hour. Not entering the arrival rates per hour is a common error in using this module. Even if you specify the time units of the dialog as minutes, the arrival rates must be per hour. The time units only refer to the durations. The duration information can also be entered using a spreadsheet like view. In addition, the values can be entered using the graphical schedule editor. With the values already entered, the graphical schedule editor displays the schedule over time as shown in Figure 10.4. If the durations and values have not already been entered, the mouse arrow can be used to draw the schedule over time. If you use the graphical editor, then you may need to fiddle with the options in order to facilitate the editing of the schedule (see Figure 10.5).

Once the schedule has been defined, the last step is to indicate in the CREATE module that it should use the SCHEDULE. This is shown in Figure 10.6 where the Time Between Arrivals Type has been specified as Schedule with the appropriate Schedule Name. The schedule as in Figure 10.4 shows that the durations add up to 24 hours so that a full day's worth of arrival rates are specified. If you simulate the system for more than 1 day, what will happen on the second day? Because the schedule has a duration, value pair for each interval, the schedule will repeat after the last duration occurs. If the schedule is specified with an infinite last duration, the value specified will be used for the rest of the simulation and the schedule will not repeat.

As indicated in this limited example, it is relatively easy to model a non-stationary Poisson arrival process in Arena™. If the arrival process to the system is non-stationary, then it is probably a good idea to vary the resources available in the system according to the

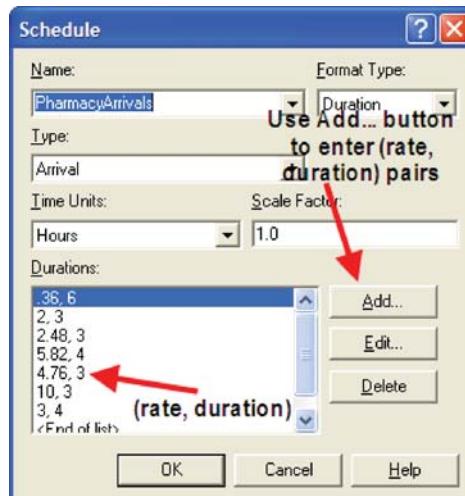


Figure 10.3 SCHEDULE dialog.

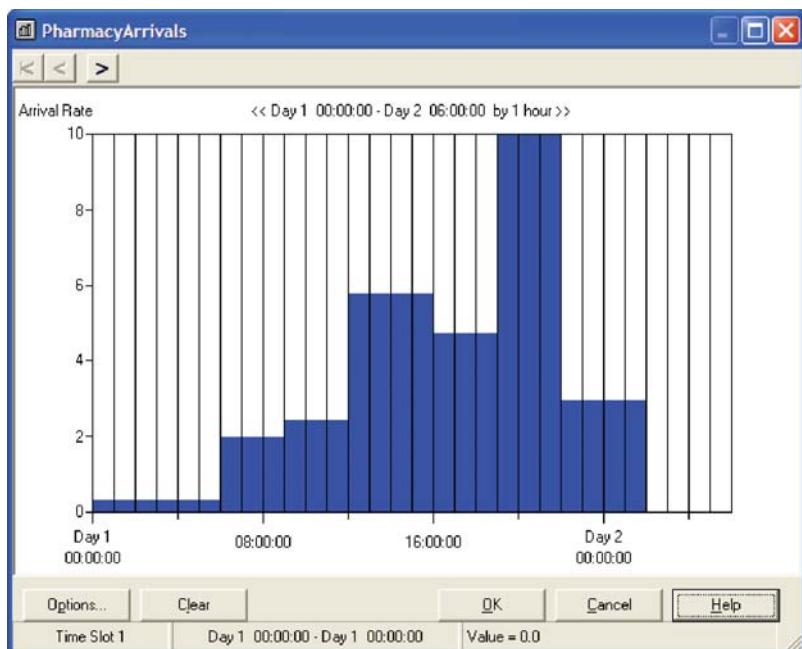


Figure 10.4 Schedule graphical editor and display.

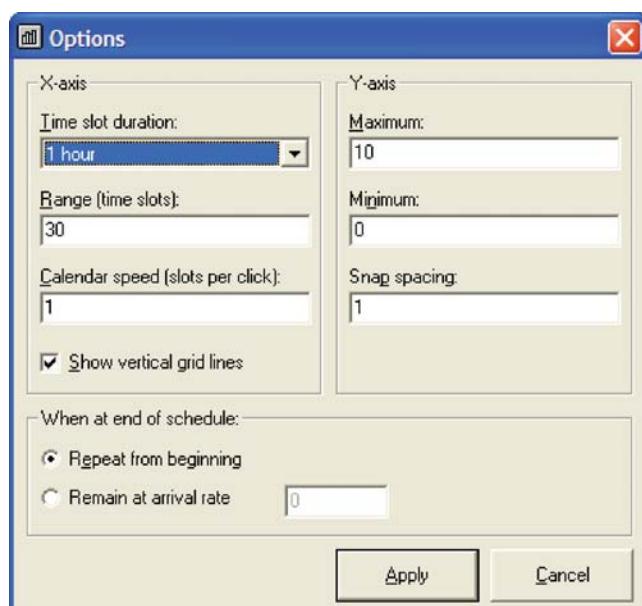
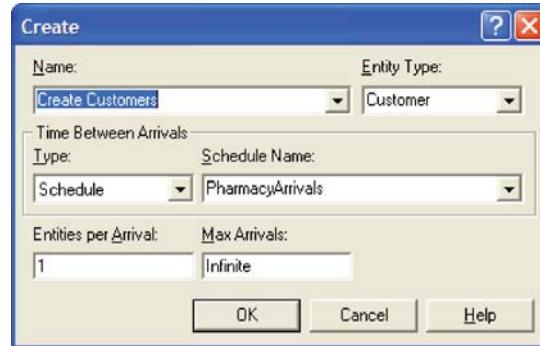


Figure 10.5 Graphical editor options.



**Figure 10.6** CREATE module with schedule type.

arrival pattern. The next section discusses how Arena<sup>TM</sup> extends the modeling of resources to include time-varying behavior as well as the ability to fail at random points in time.

### 10.3 ADVANCED RESOURCE MODELING

From previous modeling, a resource can be something that an entity uses as it flows through the system. If the required units of the resource are available for the entity, then the units are allocated to the entity. If the required units are unavailable for allocation, the entity's progress through the model stops until the required units become available. So far, the only way in which the units could become unavailable was for the units to be seized by an entity. The seizing of at least one unit of a resource causes the resource to be considered busy. A resource is considered to be in the idle state when all units are idle (no units are seized). Thus, in previous modeling, a resource could be in one of the two states: Busy or Idle. This section discusses the other two default states that Arena<sup>TM</sup> permits for resources: Failed and Inactive.

When specifying a resource, its capacity must be given. The capacity represents the maximum number of units of the resource that can be seized at any time. In previous modeling, the capacity of the resource was *fixed*. That is, the capacity did not vary as a function of time. When the capacity of a resource was set, the resource had that same capacity for the entire simulation; however, in many situations, the capacity of a resource can vary with time. For example, in the pharmacy model, you might want to have two or more pharmacists staffing the pharmacy during peak business hours. This type of capacity change is predictable and can be managed via a staffing schedule. Alternatively, the changes in capacity of a resource might be unpredictable or random. For example, a machine being used to produce products on a manufacturing line may experience random failures that require repair. During the repair time, the machine is not available for production. The situations of scheduled capacity change and random capacity change define the Inactive and Failed states within Arena<sup>TM</sup> modeling.

There are two key resource variables for understanding resources and their states.

**MR (Resource ID)** Resource capacity. MR returns the number of capacity units currently defined for the specified resource. This value can be changed by the user via resource schedule or through the use of the ALTER block.

**NR (Resource ID)** Number of busy resource units. Each time an entity seizes or pre-empts capacity units of a resource, the NR variable changes accordingly. NR is not user assignable; it is an integer value.

The index Resource ID should be the unique name of the resource or its construct number. These variables can change over time for a resource. The changing of these variables along with the possibility of a resource failing defines the possible states of a resource. These states are listed as follows:

**Idle** A resource is in the idle state when all units are idle and the resource is not failed or inactive. This state is represented by the Arena™ constant, IDLE\_RES, which evaluates to the number (-1).

**Busy** A resource is in the busy state when it has one or more busy (seized) units. This state is represented by the Arena™ constant, BUSY\_RES, which evaluates to the number (-2).

**Inactive** A resource is in the inactive state when it has zero capacity and is not failed. This state is represented by the Arena™ constant, INACTIVE\_RES, which evaluates to the number (-3).

**Failed** A resource is in the failed state when a failure is currently acting on the resource. This state is represented by the Arena™ constant, FAILED\_RES, which evaluates to the number (-4).

The special function STATE(Resource ID) indicates the current state of the resource. For example, STATE(Pharmacist) == BUSY\_RES will return true if the resource named *Pharmacist* is currently busy. Thus, the STATE() function can be used in conditional statements and within the Arena™ environment (e.g., variable animation) to check the state of a resource.

To vary the capacity of a resource, you can use the SCHEDULE module on the Basic Process panel, the Calendar Schedules builder, or the ALTER block on the Blocks panel. Only resource schedules will be presented in this text. The Calendar Schedule builder (Edit > Calendar Schedule) allows the user to define resource capacities in relation to a Gregorian calendar. For more information on this, please refer to the help system. The ALTER block can be used within the model window to invoke capacity changes based on specialized control logic. The ALTER block allows the capacity of the resource to be increased or decreased by a specified amount.

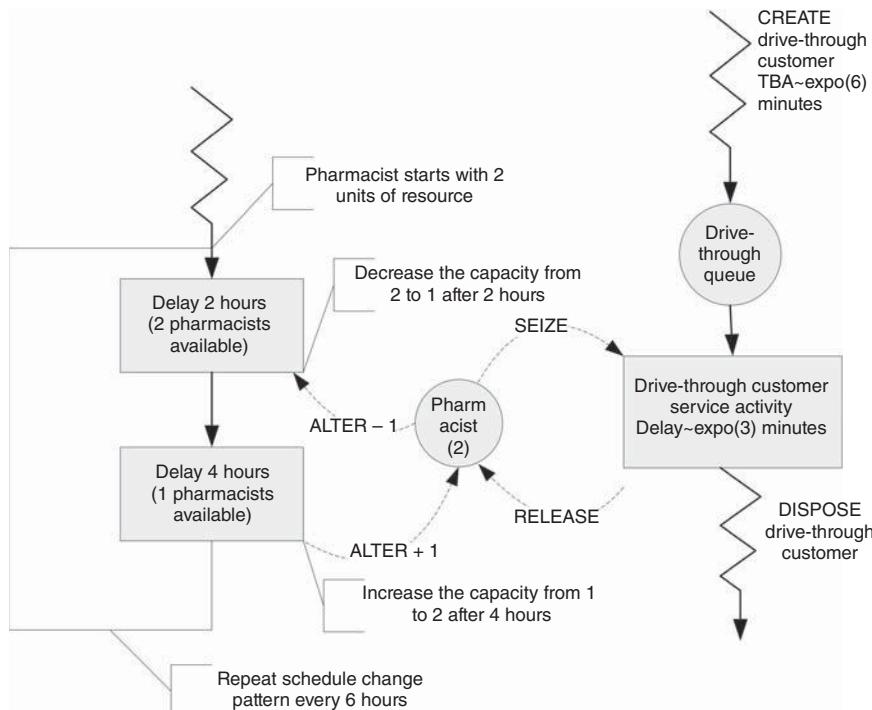
### 10.3.1 Scheduled Capacity Changes

In the pharmacy model, suppose that there were two pharmacists available at the beginning of the day to serve the drive-through window. During the first 2 hours of the day, there were two pharmacists; then during the next 4 hours, one of the pharmacists was not scheduled to work. In addition, suppose that this on-and-off pattern of availability repeated itself every 6 hours. This situation can be modeled with an activity diagram by indicating that the resource capacity is decreased or increased at the appropriate times. Figure 10.7 illustrates this concept. The diagram has been augmented with the ALTER keyword, to indicate the capacity change. Conceptually, a “schedule entity” is created in order to cycle through the pharmacist’s schedule. At the end of the first 2-hour period, the capacity is decreased to one pharmacist. Then, after the 4-hour delay, the capacity is increased back up to the two

pharmacists. This is essentially how you utilize the ALTER block within a model. Please refer to Pegden et al. [1995] and the help system for further information on the ALTER block. As can be seen in the figure, this pattern can be easily extended for multiple capacity changes and duration. This is what the SCHEDULE module does in an easier to use form.

As an example, consider the following illustration of the SCHEDULE module based on a modified version of *Smarts139-ResourceScheduleExample.doe*, an Arena™ SMART file. In this system, customers arrive according to a Poisson process with a rate of one customer per minute. The service times of the customers are distributed according to a triangular distribution with parameters (3, 4, 5) in minutes. Each arriving customer requires 1 unit of a single resource when receiving service; however, the capacity of the resource varies with time. Let us suppose that two 480-minute shifts of operation will be simulated. Assuming that the first shift starts at time 0, each shift is staffed with the same pattern, which varies according to the values shown in Table 10.3. With this information, you can easily set up the resource in to follow the staffing plan for the two shifts. This is done by first defining the schedule using the SCHEDULE module and then indicating that the resource should follow the schedule within the appropriate RESOURCE module.

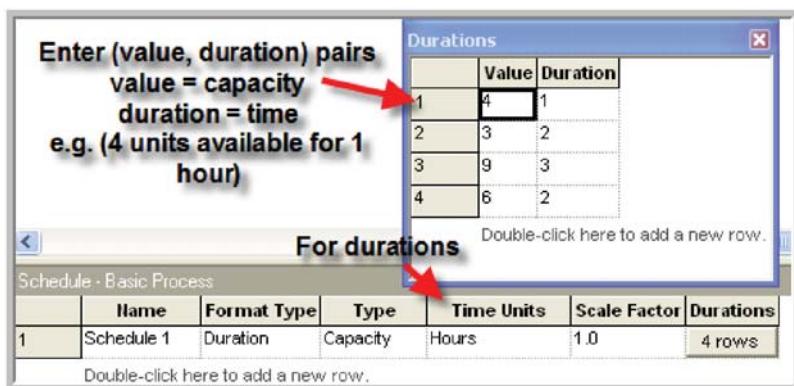
To set up the schedule, there are a number of schedule editing formats available. Figure 10.8 shows the schedule spreadsheet editor for this example. In the spreadsheet, you can enter (value, duration) pairs that represent the capacity and the duration of time that the capacity will be available. The schedule must be given a name, format type (Duration),



**Figure 10.7** Activity diagram for scheduled capacity changes.

**TABLE 10.3 Example of Staffing Schedule**

Shift	Start Time	Stop Time	# Available	Duration
1	0	60	2	60
1	60	180	3	120
1	180	360	9	180
1	360	480	6	120
2	480	540	2	60
2	540	660	3	120
2	660	840	9	180
2	840	960	6	120

**Figure 10.8** Schedule value, duration spreadsheet.

type (Capacity for resources), and time units (hours in this case). The scale factor field does not apply to capacity type schedules.

Alternatively, you can use the graphical schedule editor to define the schedule as shown in Figure 10.9. The graphical schedule editor allows you to use the mouse to draw the capacity changes. The help button describes its use. Notice that the capacity goes up and down in the figure according to the staffing plan. The Schedule dialog box is also useful in entering a schedule. The method of schedule editing is entirely your preference.

In the SCHEDULE module, only the schedule for the first 480-minute shift was entered. What happens when the duration specified by the schedule are completed? The default action is to repeat the schedule indefinitely until the simulation run length is completed. If you want a specific capacity to remain for entire run, you can enter a blank duration. This defaults the duration to infinite. The schedule will not repeat so that the capacity will remain at the specified value when the duration is invoked.

Once the schedule has been defined, you need to specify that the resource will follow the schedule by changing its Type to “Based on Schedule” as shown in Figure 10.10. In addition, you must specify the rule that the resource will use if there is a schedule change

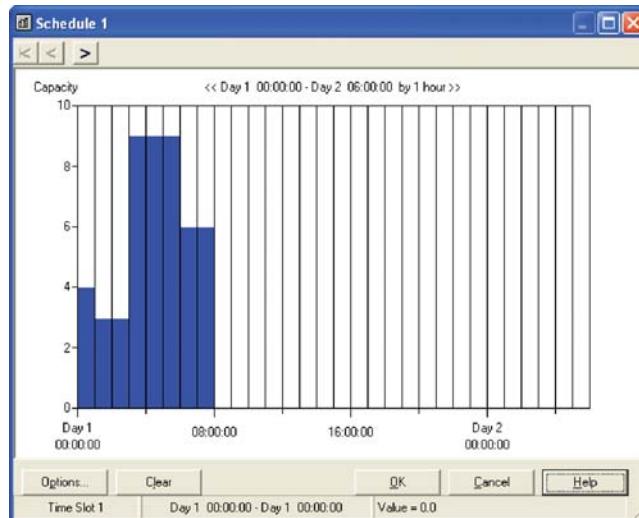


Figure 10.9 Graphical schedule editor.

and it is currently seized by entities. This is called the Schedule Rule. There are three rules available:

**Ignore** starts the time duration of the schedule change or failure immediately, but allows the busy resource to finish processing the current entity before effecting the capacity change.

**Wait** waits until the busy resource has finished processing the current entity before changing the resource capacity or starting the failure and starting the time duration of the schedule change/failure.

**Preempt** interrupts the currently processing entity, changes the resource capacity, and starts the time duration of the schedule change or failure immediately. The resource will resume processing the preempted entity as soon as the resource becomes available (after schedule change/failure).

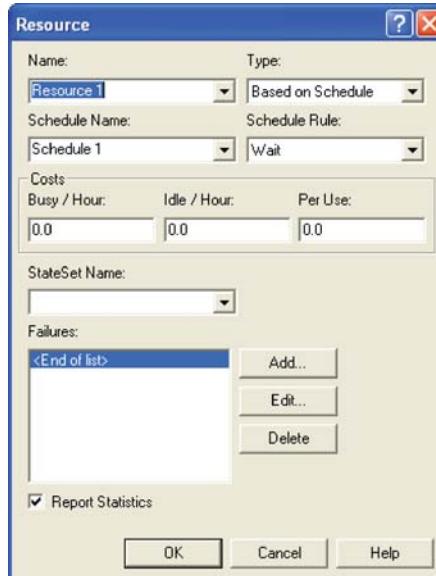
The wait rule has been used in Figure 10.10. Let us discuss a bit more on how these rules function with an example.

Let us suppose that your simulation professor has office hours throughout the day, except from 12 to 12:30 p.m. for lunch. In addition, since your professor teaches simulation using Arena™, he or she follows Arena's rules for capacity changes. What happens for each rule if you arrive at 11:55 a.m. with a question?

### Ignore

#### *Case 1*

- You arrive at 11:55 a.m. with a 10-minute question and begin service.
- At Noon, the professor gets up and hangs a Lunch in Progress sign.



**Figure 10.10** RESOURCE module dialog with scheduled capacity.

- Your professor continues to answer your question for the remaining service time of 5 minutes.
- You leave at 12:05 p.m.
- Whether or not there are any students waiting in the hallway, the professor still starts lunch.
- At 12:30 p.m., the professor finishes lunch and takes down the Lunch in Progress sign. If there were any students waiting in the hallway, they can begin service.

The net effect of case 1 is that the professor lost 5 minutes of lunch time. During the 30-minute schedule break, the professor was busy for 5 minutes and inactive for 25 minutes.

### *Case 2*

- You arrive at 11:55 a.m. with a 45-minute question and begin service.
- At Noon, the professor gets up and hangs a Lunch in Progress sign.
- Your professor continues to answer your question for the remaining service time of 40 minutes.
- At 12:30 p.m., the professor gets up and takes down the Lunch in Progress sign and continues to answer your question.
- You leave at 12:40 p.m.

The net effect of case 2 is that the professor did not get to eat lunch that day. During the 30-minute scheduled break, the professor was busy for 30 minutes.

This rule is called *Ignore* since the scheduled break may be *ignored* by the resource if the resource is busy when the break occurs. Technically, the scheduled break actually occurs and the time that was scheduled is considered as unscheduled (inactive) time.

Let us assume that the professor is at work for 8 hours each day (including the lunch break). But because of the scheduled lunch break, the total time available for useful work is 450 minutes. In case 1, the professor worked for 5 minutes more than they were scheduled. In case 2, the professor worked for 30 minutes more than they were scheduled. As will be indicated shortly, this extra work time, must be factored into how the utilization of the resource (professor) is computed. Now, let us examine what happens if the rule is *Wait*.

## Wait

### *Case 1*

- You arrive at 11:55 a.m. with a 10-minute question and begin service.
- At Noon, the professor's lunch reminder rings on his or her computer. The professor recognizes the reminder but does not act on it, yet.
- Your professor continues to answer your question for the remaining service time of 5 minutes.
- You leave at 12:05 p.m. The professor recalls the lunch reminder and hangs a *Lunch in Progress* sign. Whether or not there are any students waiting in the hallway, the professor still hangs the sign and starts a 30-minute lunch.
- At 12:35 p.m., the professor finishes lunch and takes down the *Lunch in Progress* sign. If there were any students waiting in the hallway, they can begin service.

### *Case 2*

- You arrive at 11:55 a.m. with a 45-minute question and begin service.
- At Noon, the professor's lunch reminder rings on his or her computer. The professor recognizes the reminder but does not act on it, yet.
- Your professor continues to answer your question for the remaining service time of 40 minutes.
- You leave at 12:40 p.m. The professor recalls the lunch reminder and hangs a *Lunch in Progress* sign. Whether or not there are any students waiting in the hallway, the professor still hangs the sign and starts a 30-minute lunch.
- At 1:10 p.m., the professor finishes lunch and takes down the *Lunch in Progress* sign. If there were any students waiting in the hallway, they can begin service.

The net effect of both these cases is that the professor does not miss lunch (unless the student's question takes the rest of the afternoon!). Thus, in this case, the resource will experience the scheduled break after *waiting* to complete the entity in progress. Again, in this case, the tabulation of the amount of busy time may be affected by when the rule is invoked. Now, let us consider the situation of the *Preempt* rule.

## Preempt

### Case 1

- You arrive at 11:55 a.m. with a 10-minute question and begin service.
- At Noon, the professor's lunch reminder rings on his or her computer. The professor gets up, pushes you into the corner of the office, hangs the Lunch in Progress sign and begins a 30-minute lunch. If there are any students waiting in the hallway, they continue to wait.
- You wait patiently in the corner of the office until 12:30 p.m.
- At 12:30 p.m., the professor finishes lunch, takes down the Lunch in Progress sign and tells you that you can get out of the corner. You continue your question for the remaining 5 minutes.
- At 12:35 p.m., you finally finish your 10-minute question and depart the system, wondering what the professor had to eat that was so important.

As you can see from the handling of case 1, the professor always gets the lunch break. The customer's service is preempted and resumed after the scheduled break. The result for case 2 (not shown) is essentially the same, with the student finally completing service at 12:55 p.m. While this rule may seem a bit rude in a service situation, it is quite reasonable for many situations where the service can be restarted (e.g., parts on a machine).

The example involving the professor involved a resource with 1 unit of capacity. But what happens if the resource has a capacity of more than 1 and what happens if the capacity change is more than 1. The rules work essentially the same. If the scheduled change (decrease) is less than or equal to the current number of idle units, then the rules are not invoked. If the scheduled change will require busy units, then any idle units are first taken away and then the rules are invoked. In the case of the ignore rule, the units continue serving, the inactive sign goes up, and whichever unit is released first becomes inactive first.

Now that you understand the consequences of the rules, let us run the example model. The final model including the animation is shown in Figure 10.11. When you run the model, the variable animation boxes will display the current time (in minutes), the current number of scheduled resource units (MR), the current number of busy resource units (NR), and the state of the resource (as per the resource state constants). After running the model for 960 minutes, the resource statistics results can be seen in Figure 10.12. Now you should notice something new. In all prior work, the instantaneous utilization was the same as the scheduled utilization. In this case, these quantities are not the same. The tabulation of the amount of busy time as per the scheduled rules accounts for the difference. Let us take a more detailed look at how these quantities are computed.

#### 10.3.2 Calculating Utilization

The calculation of instantaneous and scheduled utilization depends upon the two variables *NR* and *MR* for a resource. The instantaneous utilization is defined as the time-weighted

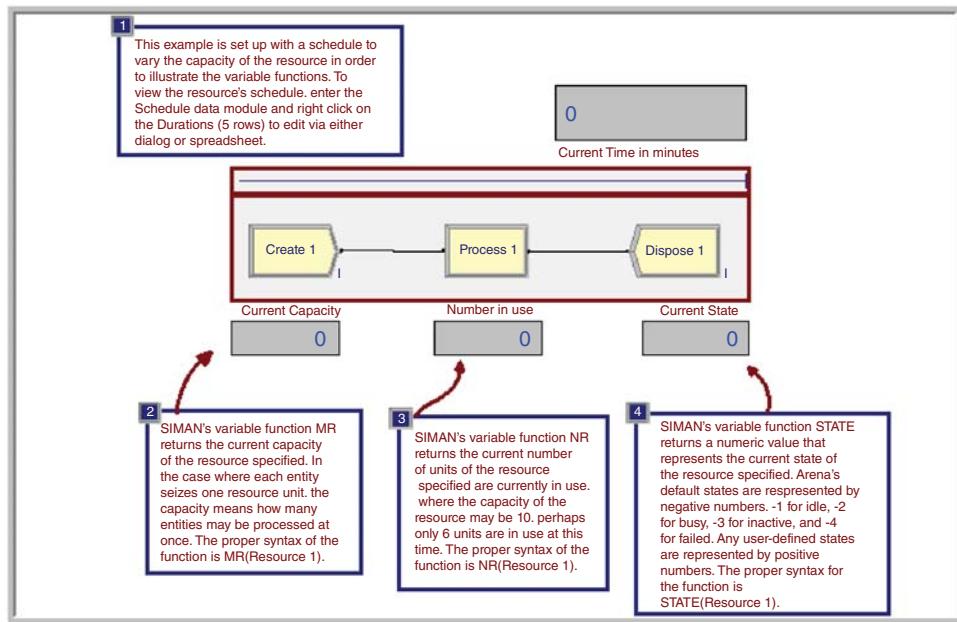


Figure 10.11 Scheduled capacity change model.

<b>Resource</b>		
<b>Usage</b>		
Instantaneous Utilization	Average	Half Width
Resource 1	0.7586	(Correlated)
Number Busy	Average	Half Width
Resource 1	4.1850	0.462899358
Number Scheduled	Average	Half Width
Resource 1	8.1154	(Insufficient)
Scheduled Utilization	Value	
Resource 1	0.6843	
Total Number Seized	Value	
Resource 1	1007.00	

Figure 10.12 Resource statistical results for scheduled capacity change example.

average of the ratio of these variables. Let  $NR(t)$  be the number of busy resource units at time  $t$ . Then, the time average number of busy resources is:

$$\overline{NR} = \frac{1}{T} \int_0^T NR(t) dt \quad (10.3)$$

Let  $MR(t)$  be the number of scheduled resource units at time  $t$ . Then, the time average number of scheduled resource units is:

$$\overline{MR} = \frac{1}{T} \int_0^T MR(t) dt \quad (10.4)$$

Now, we can define the instantaneous utilization at time  $t$  as:

$$IU(t) = \begin{cases} 0 & NR(t) = 0 \\ 1 & NR(t) \geq MR(t) \\ NR(t)/MR(t) & \text{otherwise} \end{cases} \quad (10.5)$$

Thus, the time average instantaneous utilization is:

$$\overline{IU} = \frac{1}{T} \int_0^T IU(t) dt \quad (10.6)$$

The scheduled utilization is the time average number of busy resources divided by the time average number scheduled. As can be seen in Equation 10.7, this is the same as the total time spent busy divided by the total time available for all resource units.

$$\overline{SU} = \frac{\overline{NR}}{\overline{MR}} = \frac{\frac{1}{T} \int_0^T NR(t) dt}{\frac{1}{T} \int_0^T MR(t) dt} = \frac{\int_0^T NR(t) dt}{\int_0^T MR(t) dt} \quad (10.7)$$

If  $MR(t)$  is constant, then  $\overline{IU} = \overline{SU}$ . The function RESUTIL(Resource ID) returns  $IU(t)$ . Caution should be used in interpreting  $\overline{IU}$  when  $MR(t)$  varies with time.

Now let us return to the example of the professor holding office hours. Let us suppose that the ignore option is used and consider case 2 and the 45-minute question. Let us also assume for simplicity that the professor had a take home examination due the next day and was, therefore, busy all day long. What would be the average instantaneous utilization and the scheduled utilization of the professor? Table 10.4 illustrates the calculations.

**TABLE 10.4 Example Calculation for Professor Utilization**

Time interval	Busy time	Scheduled time	$NR(t)$	$MR(t)$	$IU(t)$
8 a.m.–noon	240	240	1.0	1.0	1.0
12–12:30 p.m.	30	0	1.0	0	1.0
12:30–4 p.m.	210	210	1.0	1.0	1.0
	480	450	$\overline{NR} = 1.0$	$\overline{MR} = 450/480 = 0.9375$	$\overline{IU} = 1.0$

From Table 10.4, we can compute  $\overline{NR}$  and  $\overline{MR}$  as:

$$\overline{NR} = \frac{1}{480} \int_0^{480} 1.0 dt = \frac{480}{480} = 1.0$$

$$\overline{MR} = \frac{1}{480} \int_0^{480} MR(t) dt = \frac{1}{480} \left\{ \int_0^{240} 1.0 dt + \int_{270}^{480} 1.0 dt \right\} = 450/480 = 0.9375$$

$$\overline{IU} = \frac{1}{480} \int_0^{480} 1.0 dt = \frac{480}{480} = 1.0$$

$$\overline{SU} = \frac{\overline{NR}}{\overline{MR}} = \frac{1.0}{0.9375} = 1.06$$

Table 10.4 indicates that  $\overline{IU} = 1.0$  (or 100%) and  $\overline{SU} = 1.06$  (or 106%). Who says that professors do not give their all! Thus, with scheduled utilization, a schedule can result in the resource having its scheduled utilization higher than 100%. There is nothing wrong with this result, and if it is the case that the resource is busy for more time than it is scheduled, you would definitely want to know.

The help system has an excellent discussion of how it calculates instantaneous and scheduled utilization (as well as what they mean) in its help files, see Figure 10.13. The interested reader should search under *resource statistics: Instantaneous Utilization vs Scheduled utilization*.

### 10.3.3 Resource Failure Modeling

As mentioned, a resource has four states that it can be in: idle, busy, inactive, and failed. The SCHEDULE module is used to model planned capacity changes. The FAILURES module is used to model unplanned or random capacity changes that place the resource in the failed state. The failed state represents the situation of a breakdown for the *entire* resource. When a failure occurs for a resource, all units of the resource become unavailable and the resource is placed in the failed state. For example, imagine standing in line at a bank with three tellers. For some reason, the computer system goes down and all tellers are thus unable to perform work. This situation is modeled with the FAILURES module of the Advanced Process panel. A more common application of failure modeling occurs in manufacturing settings to model the failure and repair of production equipment.

There are two ways in which a failure can be initiated for a resource: time based and usage (count) based. The concept of time-based failures is illustrated in Figure 10.14. Notice the similarity to scheduled capacity changes. You can think of this situation as a failure “clock” ticking. The uptime delay is governed by the time to failure distribution and the downtime delay is governed by the time to repair distribution. When the time of failure occurs, the resource is placed in the failed state. If the resource has busy units, then the rules for capacity change (ignore, wait, and preempt) are invoked.

Usage-based failures occur *after* the resource has been released. Each time the resource is released, a count is incremented; when it reaches the specified number until failure value, the resource fails. Again, if the resource has busy units, the capacity change rules are invoked. A resource can have multiple failures (both time and count based) defined to govern its behavior. In the case of multiple failures that occur before the repair, the failures queue up and cause the repair times to be acted on consecutively.

Consider the following examples, all with an eight-hour period split into two four-hour shifts.

**Case A:** We have 150 workers available during an 8-hour period, and 50 of these are busy the entire 8 hours. Note that since the number scheduled is constant, both utilization statistics are identical.

**Case B:** We have 200 workers available for 4 hours, then 100 workers are available for the next 4 hours. All 100 of the workers are idle for the first 4 hours, then busy for the last 4 hours.

**Case C:** We have 200 workers available for 4 hours, then 100 workers are available for the next 4 hours. 50 of the workers are busy for the entire 8 hours.

**Case D:** We have no workers available for the first 4 hours, then 100 workers busy of the 300 available for the last 4 hours. In this last case, you could certainly advance the argument that period 1 should be excluded, and that the utilization is undefined during a period when no resources are scheduled. But Arena does not exclude selected periods (or states) from time persistent statistics (see [Frequencies Element](#)). So Arena does include this period and defines it as a utilization of 0%.

	Case A		Case B		Case C		Case D		
Scenario	4 Hour Shift	1	2	1	2	1	2	1	2
# Busy	50	50	0	100	50	50	0	100	
# Scheduled	150	150	200	100	200	100	0	300	
Utilization	33.33%	33.33%	0%	100%	25%	50%	0%	33.33%	
Total Hours Busy	$(50*8) = 400$		$((0*4)+(100*4)) = 400$		$(50*8) = 400$		$(100*4) = 400$		
Total Hours Scheduled	$(150*8) = 1200$		$((200*4)+(100*4)) = 1200$		$((200*4)+(100*4)) = 1200$		$(300*4) = 1200$		
Average Number Busy	$(50*8)/8 = 50$		$((0*4)+(100*4))/8 = 50$		$(50*8)/8 = 50$		$((0*4)+(100*4))/8 = 50$		
Average Number Scheduled	$(150*8)/8 = 150$		$((200*4)+(100*4))/8 = 150$		$((200*4)+(100*4))/8 = 150$		$((0*4)+(300*4))/8 = 150$		
Scheduled Utilization	$50/150 = 33.33\%$		$50/150 = 33.33\%$		$50/150 = 33.33\%$		$50/150 = 33.33\%$		
Instantaneous Utilization	$(33.33\%*8)/8 = 33.33\%$		$((0\%*4)+(100\%*4))/8 = 50\%$		$((25\%*4)+(50\%*4))/8 = 37.5\%$		$((0\%*4)+(33.33\%*4))/8 = 16.67\%$		

Figure 10.13 Utilization example from help files.

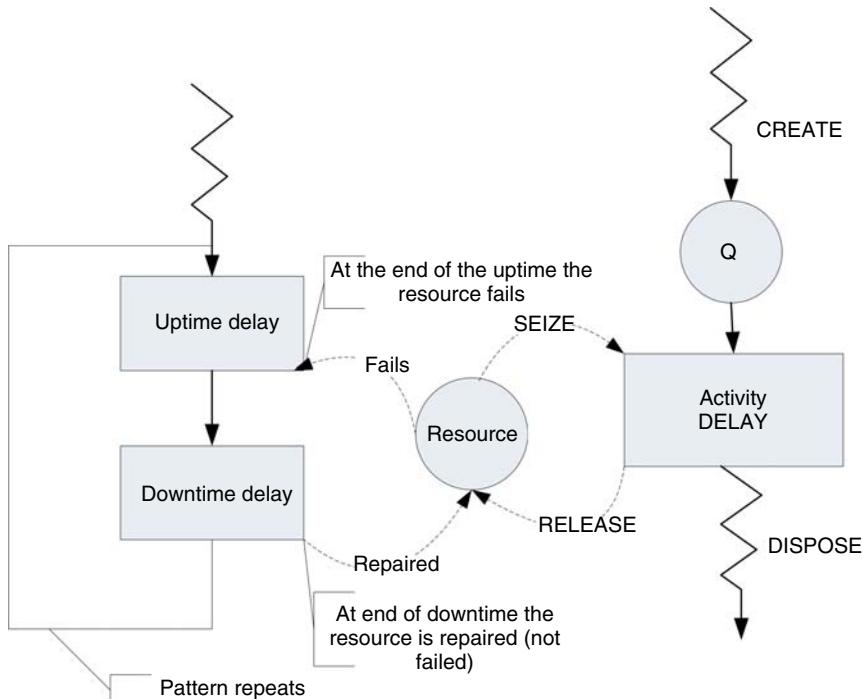


Figure 10.14 Activity diagram for failures.

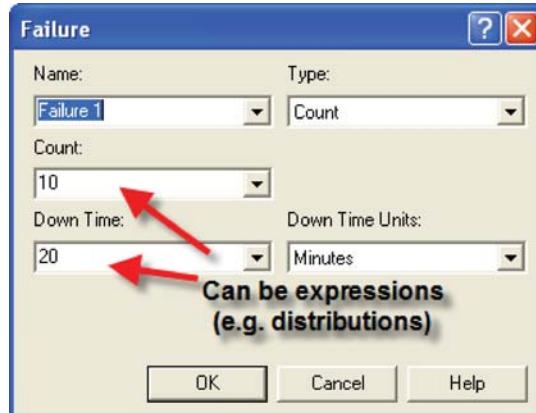


Figure 10.15 Count-based failure dialog.

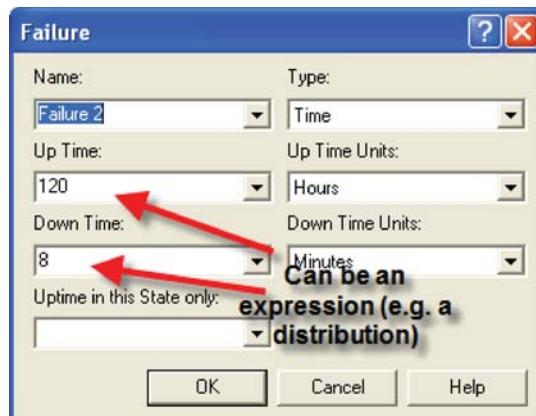


Figure 10.16 Time-based failure dialog.

Figures 10.15 and 10.16 illustrate how to define count-based and time-based failures within the FAILURES module. In both cases, the uptime, downtime, or count fields can all be expressions.

In the case of the time-based failure (Figure 10.16), the field *uptime in this State only* requires special mention. A time-based failure defines a failure event that is scheduled to occur when the failure occurs. After the failure is repaired, the event is rescheduled and the time to failure starts again. The uptime in this state field forms the basis for the time to failure. If nothing is specified, then simulation clock time is used as the basis. However, you can specify a state (from a STATESET or auto state (Idle, Busy, etc.)) to use as the basis for accumulating the time until failure. The most common situation is to choose the Busy state. In this manner, the resource only accumulates time until failure during its busy time. Figures 10.17 and 10.18 illustrate how to attach the failures to resources.

Notice that in the case of Figure 10.18, you can clearly see that a resource can have multiple failures. In addition, the same failure can be attached to different resources. These figures are all based on the file *Smarts112-TimeAndCountFailures.doe*, which accompanies this chapter. In this file, resource animation is used to indicate that the resource is in the

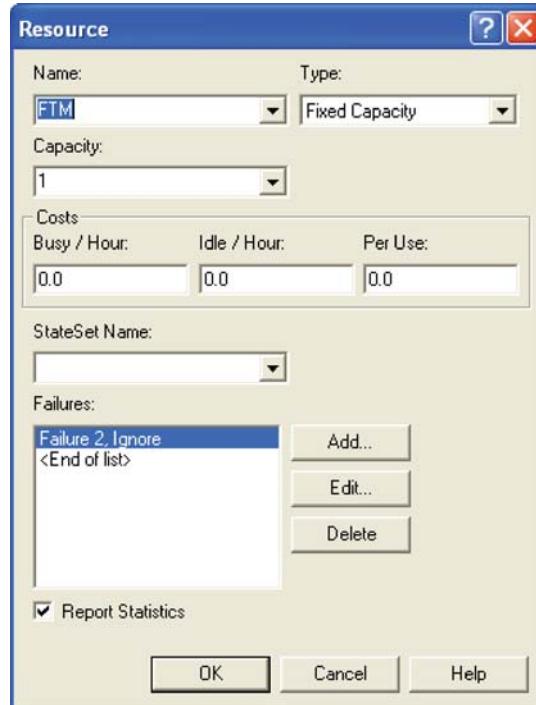


Figure 10.17 Resource with failure attached.

**Failures**

Each resource can have multiple failures. The same failure can be used on other resources.

	Failure Name	Failure Rule	Double-click here to add a new row					
	Failure Name	Failure Rule	Double-click here to add a new row					
1	Failure 1	Ignore						
2	FCNT	Fixed Capacity	1	0.0	0.0	0.0	1 rows	<input checked="" type="checkbox"/>

Figure 10.18 Resources with failures spreadsheet view.

failed state. You are encouraged to run the model. You might also look at the *Smarts123.doe* file for other ideas about displaying failures.

When resources can be in a variety of states (idle, busy, inactive, and failed), it is often useful to be able to tabulate the time (and percentage of time) spent in the states. The next section presents how Arena™ facilitates the tabulation of these quantities using the frequency option of the STATISTIC module.

## 10.4 TABULATING FREQUENCIES USING THE STATISTIC MODULE

Time-based variables within a simulation take on a particular value for a duration of time. For example, consider a variable like NQ(Queue Name) which represents the number of

entities currently waiting in the named queue. If  $NQ$  equals zero, the queue is considered empty. You might want to tabulate the time (and percentage of time) that the queue was empty. This requires that you record when the queue becomes empty and when it is not empty. From this, you can summarize the time spent in the empty state. The ratio of the time spent in the state to the total time yields the percentage of time spent in the state, and under certain conditions, this percentage can be considered the probability that the queue is empty.

To facilitate statistical collection of these quantities, the frequencies option of the STATISTIC module can be used. With the frequency option, you specify either a value or a range of values over which you want frequencies tabulated. For example, suppose that you want to know the percentage of time that the queue is empty, that it has 1 to 5 customers and 6 to 10 customers. In the first case (empty), you need to tabulate the total time for which  $NQ$  equals 0. In the second case, you need to tabulate the total time for which there were 1, 2, 3, 4, or 5 customers in the queue. This second case can be specified by a range. For the frequency statistics module, the range is specified such that it does not include the lower limit. For example, if  $LL$  represents the lower limit of the range and  $UL$  represents the upper limit of the range, the time tabulation occurs for the values in  $(LL, UL]$ . Thus, to collect the time that the queue has 1 to 5 customers, you would be able to specify a range  $(0, 5]$ . Frequencies can also be tabulated over the states of a resource, essentially using the STATE(Resource Name) variable.

The file *Smarts112-TimeAndCountFailuresWithFrequencies.doe* that accompanies this chapter illustrates the use the frequencies option. Figure 10.19 shows how to collect frequencies on a queue and a resource. In the figure, the categories for the frequency tabulation on the queue have been specified. There have been three categories defined (empty, 1 to 5, and 6 to 10). Since the empty category is defined solely on a single value, it is considered as a constant. The other two categories have been defined as ranges. The user needs to provide the value in the case of the constant or a range of values, as in the case of a range. Notice how the range for 1 to 5 has a low value specified as 0 and a high value specified as 5. This is because the lower value of the range is not included in the range. The specification of categories is similar to specifying the bins in histogram.

The following quantities will be tabulated for a frequency (as per the help files).

**FAVG (Frequency ID, Category)** Average time in category. FAVG is the average time that the frequency expression has had a value in the specified category range. FAVG equals FRQTIM divided by FCOUNT.

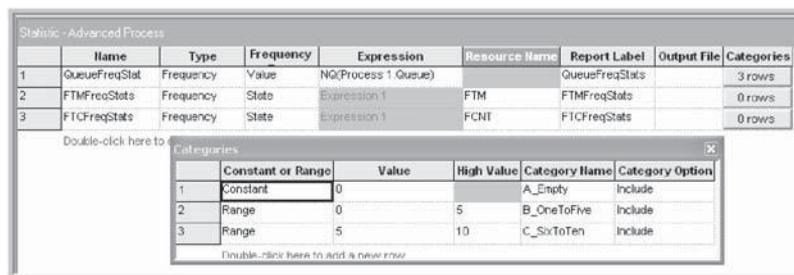


Figure 10.19 Frequency option in STATISTIC module.

**FCATS (Frequency ID)** Number of categories. FCATS returns the number of categories of a frequency, including the out-of-range category. FCATS is an integer value.

**FCOUNT (Frequency ID, Category)** Frequency category count. FCOUNT is the number of occurrences of observations for the Frequency Number of values in the Category number; it is an integer value. Only occurrences of time  $> 0$  are counted.

**FHILIM (Frequency ID, Category)** Frequency category high limit. FHILIM is the upper limit of a category range or simply the value if no range is defined for the particular Category number of Frequency Number. FHILIM is user assignable.

**FLOLIM (Frequency ID, Category)** Frequency category low limit. FLOLIM defines the lower limit of a frequency category range. Values equal to FLOLIM are not included in the Category; all values larger than FLOLIM and less than or equal to FHILIM for the category are recorded. FLOLIM is user assignable.

**FSTAND (Frequency ID, Category)** Standard category percent. FSTAND calculates the percentage of time in the specified category compared to the time in all categories.

**FRQTIM (Frequency ID, Category)** Time in category. FRQTIM stores the total time of the frequency expression value in the defined range of the Category number.

**FRESTR (Frequency ID, Category)** Restricted category percent. FRESTR calculates the percentage of time in the specified category compared to the time in all restricted categories.

**FTOT (Frequency ID)** Total frequency time. FTOT records the total amount of time that frequency statistics have been collected for the specified Frequency Number.

**FTOTR (Frequency ID)** Restricted frequency time. FTOTR records the amount of time that the specified Frequency Number has contained values in nonexcluded categories (i.e., categories that have a value in the restricted percentage column).

**FVALUE (Frequency ID)** Last recorded value. FVALUE returns the last recorded value for the specified frequency. When animating a frequency histogram, it is FVALUE, not the FAVG, which is typically displayed.

Notice that the number of occurrences or number of times that the category was observed is tallied as well as the time spent in the category. The user has the opportunity to include or exclude a particular category from the total time. This causes two types of percentages to be reported: standard and restricted. The restricted percentage is computed based on the total time that has removed any excluded categories.

Figure 10.20 illustrates the results from running the example model for 480 hours. Recall that in this model, the FTC resource subject to random failures and that the FTM resource is subject to both a schedule and random failures. From the figure, you can see that the FTC resource failed 48 times for an average time spent failed of 0.3333. This resulted in about 3.33% of the time being failed ( $480 \times 0.03333 = 15.9984$ ) hours. In the case of the FTM resource, the resource spent about 11% (or about 52.848 hours) of the total time inactive. When tabulating frequencies for a resource, the restricted percentage column automatically restricts the inactive state from its calculations. Thus, the value 93.74 for the restricted percentage busy represents 93.74% of  $480 - 52.848 = 427.151$  hours or about 400 hours. This is the same as 83.43% of 480 hours. For the frequency tabulation on the number in queue, there are four not three categories listed. The last category, OUT OF RANGE, records frequencies any time the value of the variable is not in one of the previously defined categories. Note that the categories do not have to define contiguous intervals. As indicated

<b>Replication 1</b>		Start Time:	0.00	Stop Time:	480.00	Time Units:	Hours
<b>ETCFreqStats</b>		Number Obs	Average Time	Standard Percent	Restricted Percent		
BUSY	456	0.03859380	8.42	8.42			
FAILED	48	0.3333	3.33	3.33			
IDLE	435	0.9738	88.25	88.25			
<b>FTMFreqStats</b>		Number Obs	Average Time	Standard Percent	Restricted Percent		
BUSY	343	1.1675	83.43	83.74			
FAILED	73	0.1077	1.84	1.84			
IDLE	48	0.4100	3.93	4.42			
INACTIVE	358	0.1478	11.01	--			
<b>QueueFreqStats</b>		Number Obs	Average Time	Standard Percent	Restricted Percent		
A_Empy	46	1.3034	12.49	12.88			
B_OneToFive	104	2.5785	55.87	57.00			
C_SixToTen	68	2.0211	28.63	28.52			
OUT OF RANGE	9	1.8058	3.01	--			

Figure 10.20 Results when using frequency option.

in the figure, the OUT OF RANGE category is automatically excluded from the restricted percentage column.

When using frequencies, the frequencies will be tabulated for each replication; however, frequency statistics are not automatically tabulated across replications. To tabulate across replications, you can define an OUTPUT statistic and use one of the previously discussed frequency functions (e.g., FRQTIM and FAVG). The frequency element allows finer detail on the time spent in user-defined categories. This can be very useful, especially, when analyzing the effectiveness of resource staffing schedules.

Besides the tabulation of time, the tabulation of cost is also available. Recall Figure 10.17, there are three fields related to tabulating the cost of a resource. The next section discusses the use of these fields as well as how costs can be associated with entities.

## 10.5 RESOURCE AND ENTITY COSTING

There are two types of cost-related information available within a model: resource cost and entity cost. Both of these cost models support the tabulation of costs that can support an activity-based costing (ABC) analysis within a model. ABC is a cost management system that attempts to better allocate costs to products based on the activities that they experience within the firm rather than a simple direct and indirect cost allocation. The details of ABC will not be discussed within this text; however, how Arena™ tabulates costs will be examined so that you can understand how to utilize the cost estimates available on the summary reports.

### 10.5.1 Resource Costing

Resource costing allows costs to be tabulated based on the time a resource is busy or idle.

In addition, a cost can be assigned to each time the resource is used. Let us take a look at a simple example to illustrate these concepts. In SMARTS file, *Smarts019.doe*, the processing of payment bills is modeled. The bills arrive according to a Poisson process with a mean of 1 bill every minute. The bills are first sent to a worker who handles the bill processing, which takes a processing time that is distributed according to a triangular distribution with parameters (0.5, 1.0, 1.5) minutes. Then the bills are sent to a worker who handles the

Resource - Basic Process								
Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures	Report Statistics
1 Biller	Fixed Capacity	1	7.75	7.75	.02		0 rows	<input checked="" type="checkbox"/>
2 Mailer	Fixed Capacity	1	5.15	5.15	.02		0 rows	<input checked="" type="checkbox"/>

Double-click here to add a new row.

Figure 10.21 Specifying resource costs.

mailing of the bills. The processing time for the mailing is also distributed according to a triangular distribution with parameters (0.5, 1.0, 1.5) minutes. Both workers are paid an hourly wage regardless of whether or not they are busy. The bill processing worker is paid \$7.75 per hour and the mailing worker is paid \$5.15 per hour. In addition, the workers are paid an additional 2 cents for each bill that they process. Management would like to tabulate the cost of this processing over an 8-hour day.

It should be clear that you can get the wage cost by simply multiplying the hourly wage by 8 hours because the workers are paid regardless of whether they are busy or idle during the period. The number of bills processed will be random and thus the total cost must be simulated. In the case of the workers getting paid differently if they are busy or idle, then the model can facilitate this calculation as well. In this example, Arena™ can do all the calculations if the costs within the RESOURCE module are specified.

The model is very simple (CREATE, PROCESS, PROCESS, DISPOSE). You should refer to the *Smarts019.doe* file for the details of the various dialog boxes. To implement the resource costing, you must specify the RESOURCE costs as per Figure 10.21.

On the Run Setup dialog, make sure that the Costing check box is enabled on the Project Parameters tab. This will ensure that the statistical reports include the cost reports. Arena™ will tabulate the costs and present a summary across all resources and allow you to drill down to get specific cost information for each resource for each category (busy, idle, and usage) as shown in Figure 10.22.

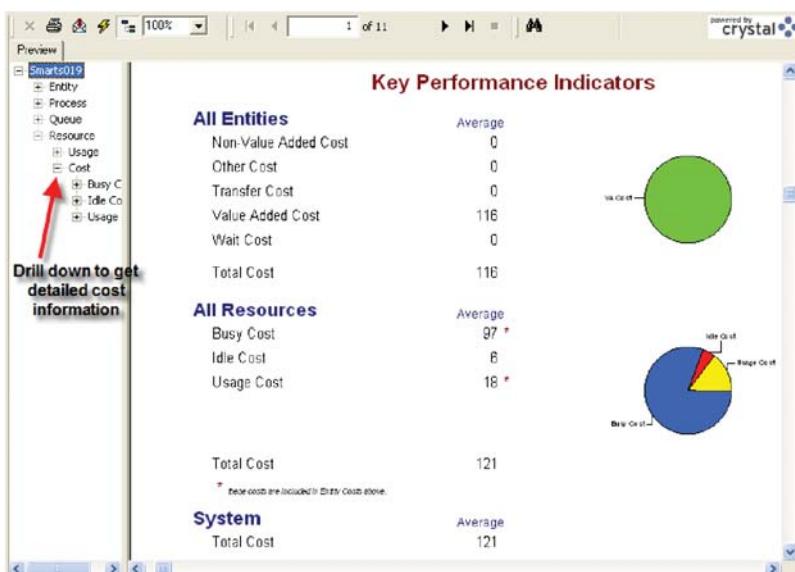


Figure 10.22 Resource cost summary.

**TABLE 10.5 Tabulation of Resource Costs**

Resource Cost	Dollars per Hour	Hours	Cost	Totals
Biller	7.75	8	62.0	
Mailer	5.15	8	41.2	
				103.2
Usage Cost	Dollar per Usage	# Uses	Cost	
Biller	0.02	456	9.12	
Mailer	0.02	456	9.12	
				18.24
			Total Cost	121.44

**TABLE 10.6 Tabulation of Busy and Idle Costs**

Resource	Utilization	Time Busy	Dollars per Hour	Cost	Total
Biller	0.9374	7.50	7.75	58.12	
Mailer	0.9561	7.65	5.15	39.39	
					97.51
	Idle	Time Idle	Dollars per Hour	Cost	
Biller	0.0626	0.50	7.75	3.88	
Mailer	0.0439	0.35	5.15	1.81	
					5.69
			Total cost	103.20	

Tables 10.5 and 10.6 indicate how the costs are tabulated from Arena's statistical reports for the resources. In Table 10.5, the number of uses is known because there were 435 entities (bills) processed by the system. In Table 10.6, the utilization from the resource statistics was used to estimate the percentage of time busy and idle. From this, the amount of time can be calculated and from that, the cost. Arena™ tabulates the actual amount of time in these categories. The value-added cost calculation will be discussed shortly.

To contrast this, consider *Smarts049.doe* in which the billing worker and mailing workers follow a schedule as shown in Figures 10.23 and 10.24. Because the resources have less time available to be busy or idle the costs are less for this model as shown in Figure 10.25.

Resource - Basic Process							
	Name	Type	Schedule Name	Schedule Rule	Busy / Hour	Idle / Hour	Per Use
1	Biller	Based on Schedule	Schedule 1	Wait	7.75	7.75	.02
2	Mailer	Based on Schedule	Schedule 1	Wait	5.15	5.15	.02
Double-click here to add a new row.							

**Figure 10.23** Smarts049 with resources based on schedule.

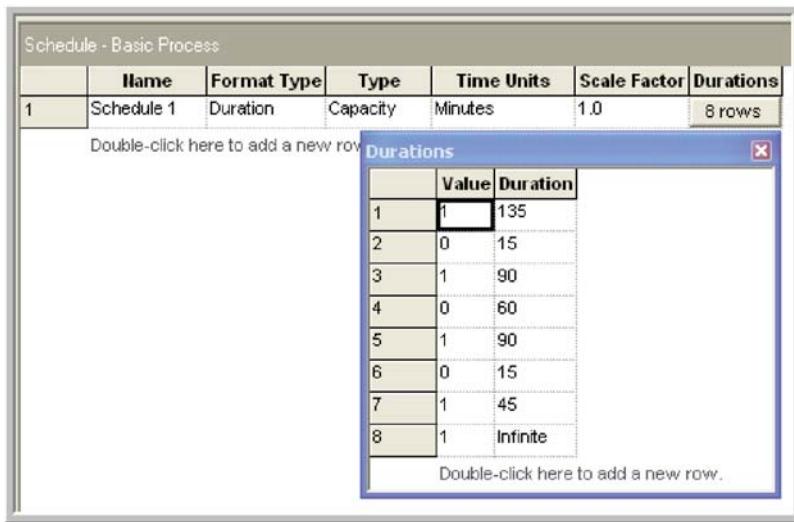


Figure 10.24 Schedule for Smarts049.

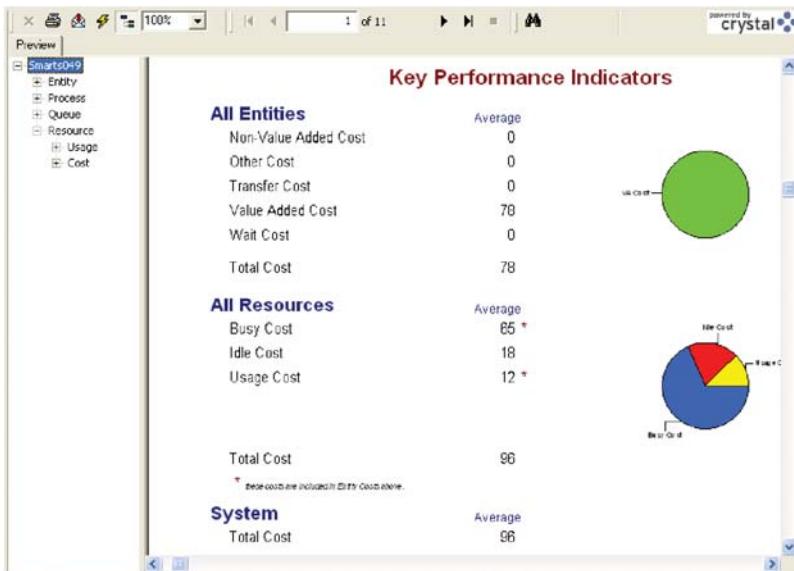


Figure 10.25 Summary costs for resources following a schedule.

In this case, the workers are not paid when they are not scheduled. As can be seen in Figure 10.25, Arena™ also tabulates costs for the entities. Let us take a look at another example to examine how entity costs are tabulated.

### 10.5.2 Entity Costing

Entity costing is slightly more complex than resource costing. Arena™ assigns entity costs into five different activity categories.

**Waiting Time/Cost** Wait time is any time designated as Wait. Waiting time in queues is by default allocated as waiting time. Waiting cost is the cost associated with an entity's designated wait time. This cost includes the value of the time the entity spends in the waiting state and the value of the resources that are held by the entity during the time.

**Value-Added Time/Cost** Value-added time is any time designated as Value added.

Value-added time directly contributes value to the entity during the activity. For example, the insertion of components on a printed circuit board adds value to the final product. Value-added cost is the cost associated with an entity's designated value-added time. This cost includes the value of the time the entity spends in the value-added activity and the value of the resources that are held by the entity during the time.

**Nonvalue-added time/cost** Nonvalue-added time is any time designated as nonvalue added. Nonvalue-added time does not directly contribute value to the entity during the activity. For example, the preparation of the materials needed to insert the components on the printed circuit board (e.g., organizing them for insertion) can be considered as nonvalue added. The designation of nonvalue-added time can be subjective. As a rule, if the time could be reduced or eliminated and efficiency increased without additional cost, then the time can be considered as nonvalue added. Nonvalue-added cost is the cost associated with an entity's designated nonvalue-added time. This cost includes the value of the time the entity spends in nonvalue-added activity and the value of the resources that are held by the entity during the time.

**Transfer Time/Cost** Transfer time can be considered a special case of nonvalue-added time in which the entity experiences a transfer. By default, all time spent using material handling devices from the Advanced Transfer panel (such as a conveyor or transporter) is specified as Transfer time. Transfer cost is the cost associated with an entity's designated transfer time. This cost includes the value of the time the entity spends in the designated transfer activity and the value of the resources that are held by the entity during the time.

**Other Time/Cost** This category is a catch-all for any activities that do not naturally fit the other four categories.

As mentioned, Arena<sup>TM</sup> designates wait and transfer time automatically based on the constructs being used. In the case of the transfer time category, you can also designate specific delays as transfer time, even if a material handling device is not used. In the modeling, it is incumbent on the modeler to properly designate the various activities within the model; otherwise, the resulting cost tabulations will be meaningless. Thus, Arena<sup>TM</sup> does not do the cost modeling for you; however, it tabulates the costs automatically for you. Do not forget the golden rule of computing: "Garbage in = Garbage out." When you want to use Arena<sup>TM</sup> for cost modeling, you need to *carefully* specify the cost elements within the *entire* model. Let us take a closer look at how Arena<sup>TM</sup> tabulates the costs.

Entity cost modeling begins by specifying the cost rates for the types of entities in the ENTITY module. In addition, to get a meaningful allocation of the resource cost for the entity, you need to specify the costs of using the resource as per the RESOURCE module. As can be seen in Figure 10.27, the ENTITY module allows the user to specify the holding cost of the entity as well as initial costs for each of the five categories. Let us ignore the initial costs for a moment and examine the meaning of the holding cost per hour field. The holding cost per hour represents the cost of processing the entity *anywhere* in the system.

This is the dollar value of having the entity in the system expressed as a time rate. For example, in an inventory model, you might consider this the cost of holding 1 unit of the item in the system for 1 hour. This cost can be difficult to estimate. It is typically based on a cost of capital argument; see Silver et al. [1998] for more on estimating this value. Provided the holding cost per hour for the entity is available and the resource costs are specified for the resources used by the entity, Arena™ can tabulate the costs.

Let us consider tabulating the value-added cost for an entity. As an entity moves through the model, it will experience activities designated as value added. Let  $n$  be the number of value-added activity periods experienced by the entity while in the system. Let  $VAT_i$  be the value-added time for period  $i$  and  $h$  be the holding cost rate for the entity's type. While the entity is experiencing the activity, it may be using various resources. Let  $r_i$  be the number of resources used by the entity in activity period  $i$ , and let  $b_j$  be the busy cost per hour for the  $j$ th resource held by the entity in period  $i$ . Let  $u_j$  be the usage cost associated with the  $j$ th resource used during the activity. Thus, the value-added cost,  $VAC_i$ , for the  $i$ th period is:

$$\begin{aligned} VAC_i &= h \times VAT_i + \left( \sum_{j \in R_i} b_j \right) \times VAT_i + \sum_{j \in R_i} u_j \\ &= \left( h + \sum_{j \in R_i} b_j \right) \times VAT_i + \sum_{j \in R_i} u_j \end{aligned}$$

The quantity,  $\sum_{j \in R_i} b_j$ , is called the resource cost rate for period  $i$ . Thus, the total value-added cost,  $TVAC$ , for the entity during the simulation is:

$$TVAC = \sum_{i=1}^n VAC_i \quad (10.8)$$

The costs for the other categories are computed in a similar manner by noting the number of periods for the category and the associated time spent in the category by period. These costs are then totaled for all the entities of a given type.

The initial costs as specified in the ENTITY module are treated in a special manner by Arena's cost reports. Arena's help system has this to say about the initial costs:

The initial VA cost, NVA cost, waiting cost, transfer cost, and other cost values specified in this module are automatically assigned to the entity's cost attributes when the entity is created. These initial costs are not included in the system summary statistics displayed in the Category Overview and Category by Replication reports. These costs are considered to have been incurred outside the system and are not included in any of the All Entities Cost or the Total System Cost. These initial costs are, however, included in the cost statistics by entity type.

To illustrate entity cost modeling, consider SMARTS file *Smarts047.doe*. In this model, contracts arrive according to a Poisson process with a mean rate of 1 contract per hour. There are two value-added processes on the contract: an addendum is added to the contract and a notary signs the contract. These processes each take TRIA(0.5, 1, 1.5) hours to complete. The contract processor has an \$8 per hour busy/idle cost. The notary is paid on a per usage basis of \$10 per use. The holding cost is \$1 per hour for the contracts. These values are

shown in Figures 10.26 and 10.27. Figure 10.28 illustrates how to allocate the value-added time within the contract addendum process.

By running this model with 1 entity, you can more easily see how the value-added costs are tabulated. If you run the model, the value-added cost for 1 entity is about \$19.

Figure 10.29 indicates that the value-added processing time for the contract at the addendum process was 55.4317 minutes. This can be converted to hours as shown in Table 10.7. Then the cost per hour for the entity in the system (holding cost plus resource cost) is tabulated (e.g.,  $\$1 + \$8 = \$9$ ). This is multiplied by the value-added time in hours to get the cost of the process. If the resource for the process has a resource cost, then it must be included as shown in Table 10.7 and as explained in the equation for  $VAC_i$ .

To get the cost for the entire simulation, this calculation would be done for each contract entity for every value-added activity experienced by the entity. In the case of the single entity, the total cost of \$19.16 for the addendum and notary activities is shown in Table 10.7.

Again, when using the costing constructs within your models you should be extra diligent in entering the information and understanding the effect of specifying entity types. The following issues should be thought about with care:

- Make sure that only those DISPOSE modules for which you want entity statistics tabulated have their entity statistic check box enabled.
- Make sure that you use the ENTITY module and specify the type of entity to create within your CREATE modules.

Resource - Basic Process						
	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use
1	Processor	Fixed Capacity	1	8.00	8.00	0.0
2	Notary	Fixed Capacity	1	0.0	0.0	10.00

Double-click here to add a new row.

Figure 10.26 Resource costs for Smarts047.doe.

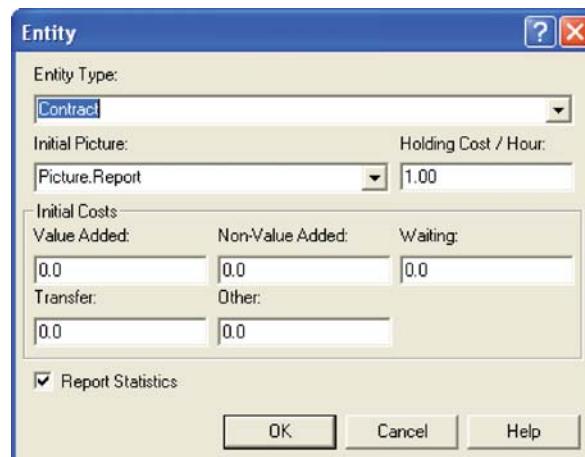


Figure 10.27 ENTITY Module Costing Terms

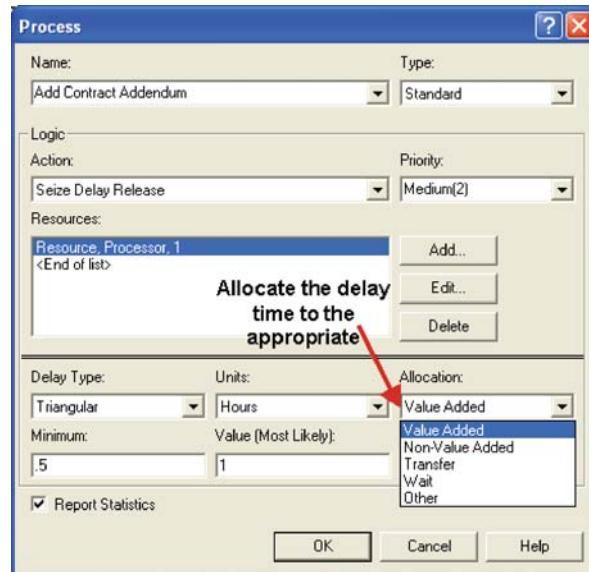


Figure 10.28 Allocating the value-added time.

Process	
Time per Entity	
VA Time Per Entity	Average
Add Contract Addendum	55.4317
Sign and Notarize Contract	50.6456
Wait Time Per Entity	Average
Add Contract Addendum	0.00
Sign and Notarize Contract	0.00
Total Time Per Entity	Average
Add Contract Addendum	55.4317
Sign and Notarize Contract	50.6456

Figure 10.29 Processing time.

- Carefully specify the allocation for your activities. For example, if you want an entity's value-added cost, then you must allocate to value added for every pertinent value-added activity that the entity may experience within the model.
- Make sure to read carefully how the BATCH and SEPARATE modules handle the assignment of entity attributes so that the proper attribute values are carried by the entities. Specify the attribute assignment criteria that best represents your situation.

**TABLE 10.7 Tabulation of Entity Cost**

Process	VAT	VAT	Holding Cost	Resource Cost	Cost	Usage Cost (\$)	Total Cost (\$)
	(Minutes)	(Hours)	(\$ per Hour)	(\$ per Hour)	(\$ per Hour)		
Addendum	55.4317	0.9238617	1.00	8.00	9.00	8.31	0.00
Notary	50.6456	0.8440933	1.00	0.00	1.00	0.84	10.00
							19.16

- Read carefully how Arena™ handles PROCESS submodels and regular submodels in terms of costing tabulation if you use submodels. See *Submodel Processing* within the help system.

Remember also that you do not have to use the built-in costing models. You can specify your cost tabulations directly within the model and tabulate only the things that you need. In other words, you can do it yourself.

The next section outlines a number of other constructs within Arena™ that can enhance your models.

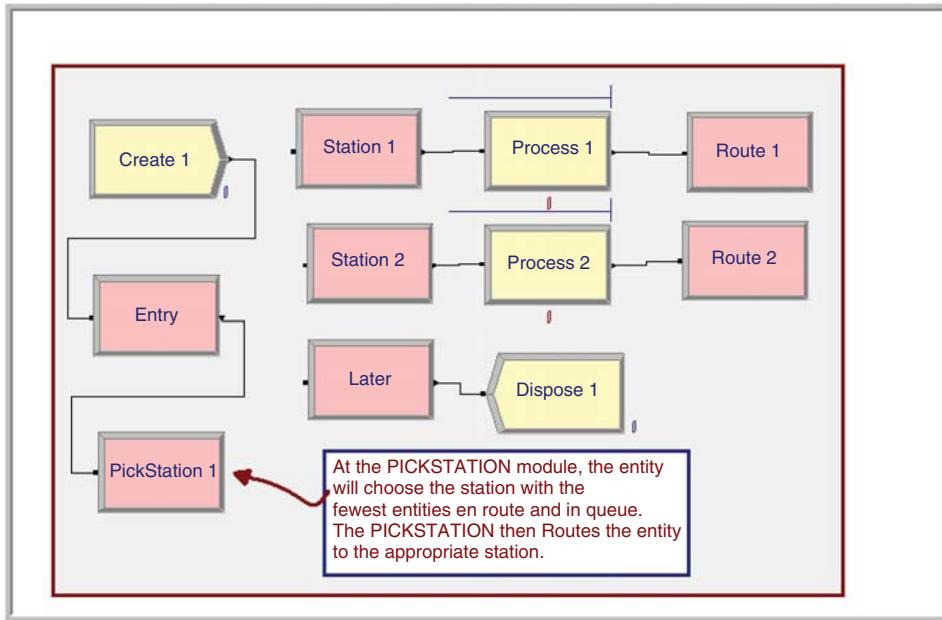
## 10.6 MISCELLANEOUS MODELING CONCEPTS

This section discusses a number of additional constructs available within Arena™ that facilitate common modeling situations that have not been previously described. The first topic allows for finer control of the routing of entities to stations for processing. The second situation involves improving the scalability of your models by making stations generic. Lastly, the use of the active entity to pick up and drop off other entities will be presented. This allows for another way to form groups of entities that travel together that is different from the functionality available in the BATCH module.

### 10.6.1 Picking Between Stations

Imagine that you have just finished your grocery shopping and that you are heading toward the check-out area. You look ahead and scan the check-out stations in order to pick what you think will be the line that will get your check-out process completed the fastest. This is a very common situation in many modeling instances: the entity is faced with selecting from a number of available stations based on the state of the system. While this situation can be handled with the use of DECIDE modules, the implementation of many criteria to check can be tedious (and error prone). In addition, there are often a set of common criteria to check (e.g., shortest line and least busy resource). Because of this, Arena™ has available the PICKSTATION module on the Advanced Transfer panel. The PICKSTATION module combines the ability to check conditions prior to transferring the entity out of a station.

SMARTS file *Smarts113.doe* represents a good example for this situation. In the model, entities are created according to a Poisson process with a rate of 1 every minute. The arriving entities then must choose between two stations for service. In this case, their choice should be based on the number of entities en route to the stations and the current number of entities



**Figure 10.30** An example of PICKSTATION module.

waiting at the stations. After receiving processing at the stations, the entities then depart the system.

As can be seen in Figure 10.30, the entity is created and then immediately enters the Entry station. From the Entry station, the entity will be routed via the PICKSTATION module to one of the two processing stations. The processing is modeled with the classic SEIZE, DELAY, RELEASE logic. After the processing, the entity is routed to the Later (exit) station where it is disposed. The only new modeling construct required for this situation is the PICKSTATION module.

The PICKSTATION module is very similar to a ROUTE module as shown in Figure 10.31. The top portion of the module defines the criteria by which the stations will be selected. The bottom portion of the module defines the transfer out mechanism. In the figure, the module has been set to select based on the minimum of the number in queue and the number en route to the station. The Test Condition field allows the user to pick according to the minimum or the maximum. The Selection Based On section specifies the constructs that will be included in the selection criterion. The user must provide the list of stations and the relevant construct to test at each station (e.g., the process queue). The selection of the construct is dependent on criteria that were selected in the Selection Based On area.

Figure 10.32 illustrates the dialog box for adding a station to the list when all criteria have been selected. Notice that the specification is based on the station. In the figure, the user can select the station, a queue at the station, a resource at the station, or a general expression to be evaluated based on the station. Arena™ scans the list to find the station that has the minimum (or maximum) according to the criteria selected. In the case of ties, Arena™ will pick the station that is listed first according to the Stations list. After the station has been selected, the entity will be routed according to the standard route options

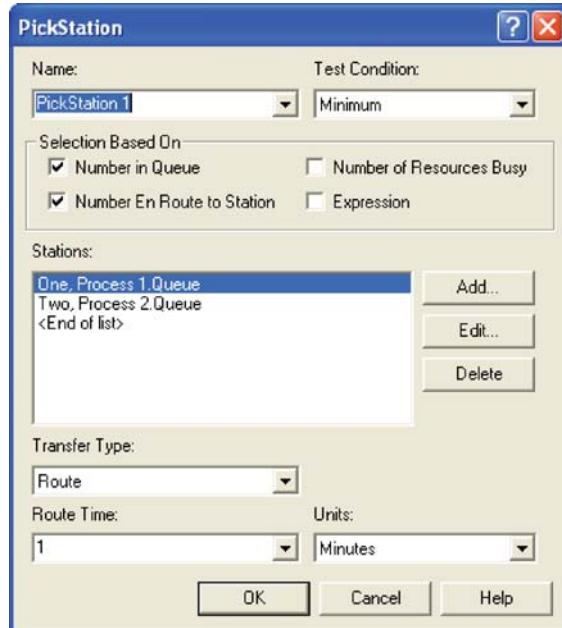


Figure 10.31 PICKSTATION module.



Figure 10.32 Station criteria specification.

(transport, route, convey, and connect). In the case of the connect option, the user is given the option to save the picked station in an attribute, which can then be used to specify the station in a transfer module (e.g., CONVEY). In addition, since the choice of transport or convey requires that the entity has control over a transporter or has accessed the conveyor, the entity must first have used the proper modules to gain control of the material handling construct (e.g., REQUEST) prior to entering the PICKSTATION.

As you can see, the specification and operation of the PICKSTATION module is relatively straightforward. The only difficulty lies in properly using the Expression option.

(a) Schedule for resource 1:

	Value	Duration
1	0	10
2	1	50

Double-click here to add

(b) Schedule for resource 2:

	Value	Duration
1	1	30
2	0	10
3	1	20

**Figure 10.33** Schedules for PICKSTATION extension. (a) Schedule for resource 1 and (b) schedule for resource 2.

Let us consider a modification to the example to illustrate the use of an expression. Let us suppose that the two resources within the example follow a schedule such that every hour the resources take a 10-minute break, with the first resource taking the break at the beginning of the hour and the second resource taking a break at the middle of the hour. This ensures that both resources are not on break at the same time. In this situation, arriving customers would not want to choose the station with the resource on break; however, if the station is picked based solely on the number in queue and the number en route, the entities may be sent to the station on break. This is because the number in queue and the number en route will be small (if not zero) when the resource is on break. Therefore, some way is needed to ensure that the station on break is not chosen during the break time. This can be done by testing to see if the resource is inactive. If it is inactive, the criteria can be multiplied by a large number so that it will not be chosen.

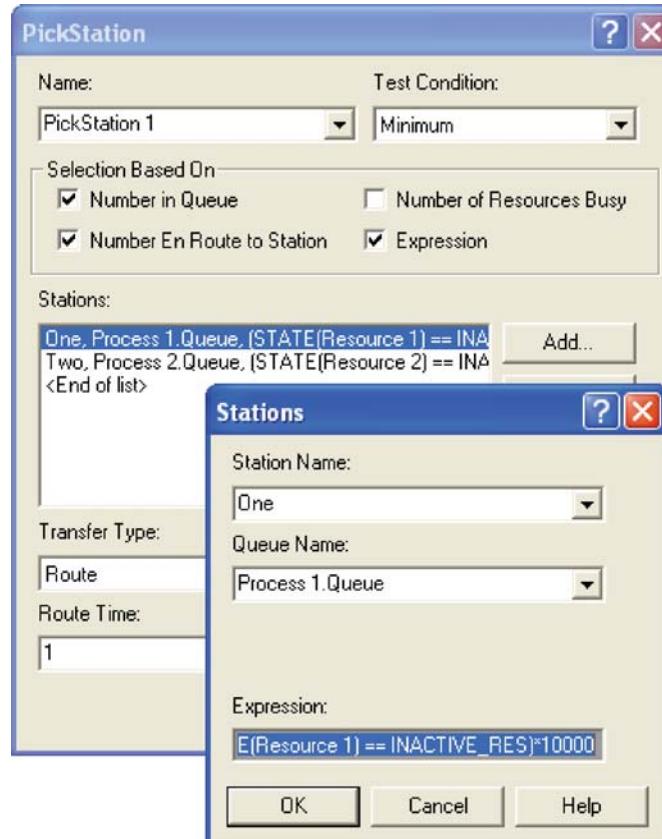
To implement this idea, two schedules were created, one for each resource to follow. Figure 10.33 shows the two schedules. In the schedule for resource 1, the capacity is 0 for the first 10 minutes and then is 1 for the last 50 minutes of the hour. The schedule then repeats. For the second resource's schedule, the resource is available for 30 minutes and then the break starts and lasts for 10 minutes. The resource is then available for the rest of the hour. This allows the schedule to repeat hourly.

Now the PICKSTATION module needs to be adjusted so that the entity does not pick the station with the resource that is on break. This implementation can be accomplished with the expression:  $(\text{STATE}(\text{resource name}) == \text{INACTIVE\_RES}) * vBigNumber$ . In this case, if the named resource's state is inactive, the Boolean expression yields 1 for true. Then the 1 is multiplied by a big number (e.g., 10,000), which would yield the value 10,000 if the state is inactive. Since the station with minimum criteria is to be selected, this will penalize any station that is currently inactive. The implementation of this in the PICKSTATION module is shown in Figure 10.34. If you run the model, *Smarts113-PickStation-ResourceStaffing.doe*, that accompanies this chapter, you can see that the entity does not pick the station that is on break.

In many models (including this one), the station logic is essentially the same, SEIZE, DELAY, RELEASE. The next section considers the possibility of replacing many stations with a generic station that can handle any entity intended for a number of stations.

### 10.6.2 Generic Station Modeling

A generic station is a station (and its accompanying modules) that can handle the processing for any number of designated stations. To accomplish this, sets must be used to their fullest.



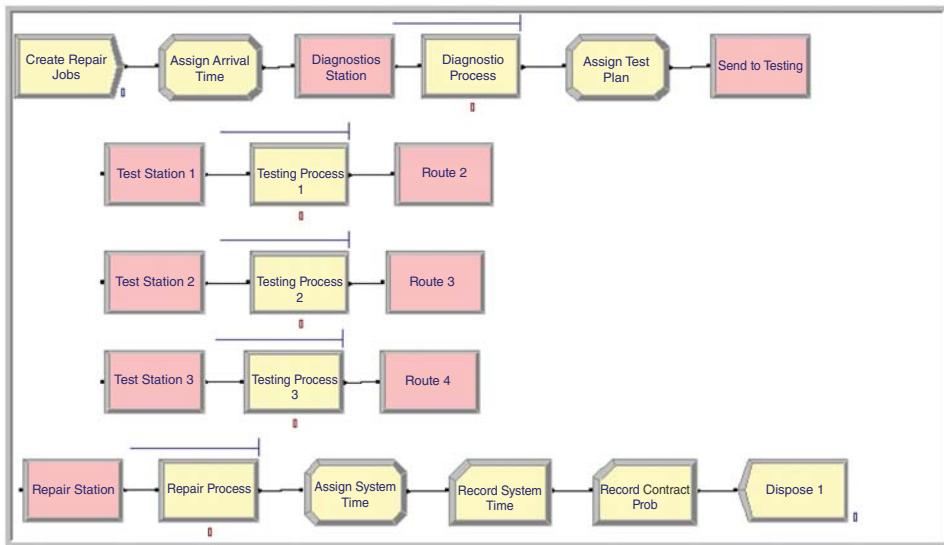
**Figure 10.34** Preventing a station from being chosen while inactive.

Generic stations allow a series of commonly repeating modules to be replaced by a set of modules, sort of like a subroutine. However, the analogy to subroutines is not very good because of the way Arena™ handles variables and attributes. The trick to convert a series of modules to generic stations is to properly index into sets. An example is probably the best way to demonstrate this idea.

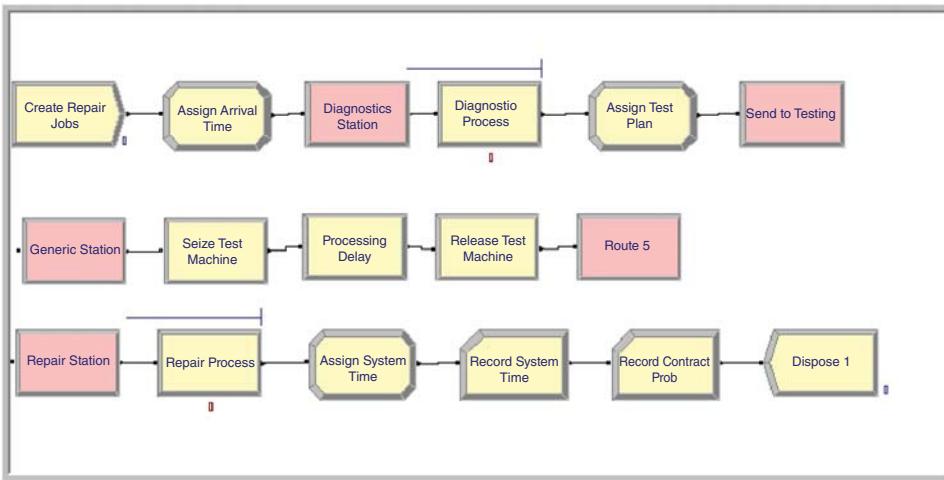
Let us reconsider the test and repair shop from Chapter 8 shown in Figure 10.35. Notice that the three testing stations in the middle of the model all have the same structure. In fact, the only difference between them is that the processing occurs at different stations. In addition, the parts are routed via sequences and have their processing times assigned within the SEQUENCE module. These two facts will make it very easy to replace those nine modules with four modules that can handle any part intended for any of the test stations.

Figure 10.36 shows the generic version of the test and repair model. As shown in the figure, the testing stations have been replaced by the station name Generic Station. Notice that the generic modeling is STATION, SEIZE, DELAY, RELEASE, ROUTE. The real trick of this model begins with set modeling.

Three sets must be defined for this model: a set to hold the test stations, a set to hold the test machine resources, and a set to hold the processing queues for the test machines.



**Figure 10.35** Test and repair model.



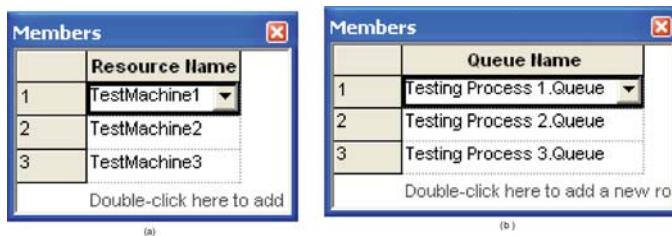
**Figure 10.36** Generic test and repair model.

There are two ways to define a set of stations: (i) using the Advanced Set module or (ii) using the set option within the STATION module. Since a generic station is being built, it is a bit more convenient to use the set option within the STATION module.

Figure 10.37 illustrates the generic station module. Notice how the station type field has been specified as “Set.” This allows a set to be defined to hold the stations to be represented by this station set. The station set members for the test stations have all been added. Arena™ will use this one module to handle any entity that is transferred to any of the listed stations. A very important issue in this modeling is the order of the stations listed. In the other sets



**Figure 10.37** Generic STATION module with set option.



**Figure 10.38** Resource and Queue sets for generic model. (a) Test machine set and (b) test machine queue set.

that will be used, you need to make sure that the resources and queues associated with the stations are listed in the same order.

When an entity is sent to one of the stations listed, which station the generic station is currently representing must be known. This is accomplished with the Save Attribute. The saved attribute will record the *index* of the station in its set.

That is important enough to repeat. It does not hold the station selected. It holds the index for the station in the station set. This index can be used to reference into a queue set and a resource set to make the processing generic. Figure 10.38 illustrates the resource and queue sets needed for the model. The resource set is defined using the SET module on the Basic Process panel, and the queue set is defined using the Advanced SET module on the Advanced Process panel. With these sets defined, they can be used in the SEIZE and RELEASE modules.

In the original model, a PROCESS module was used to represent the testing processes. In this generic model, the PROCESS module will be replaced with the SEIZE, DELAY, and RELEASE modules from the Advanced Process panel. The primary reason for doing this is so that the queue set can be accessed when seizing the resource.

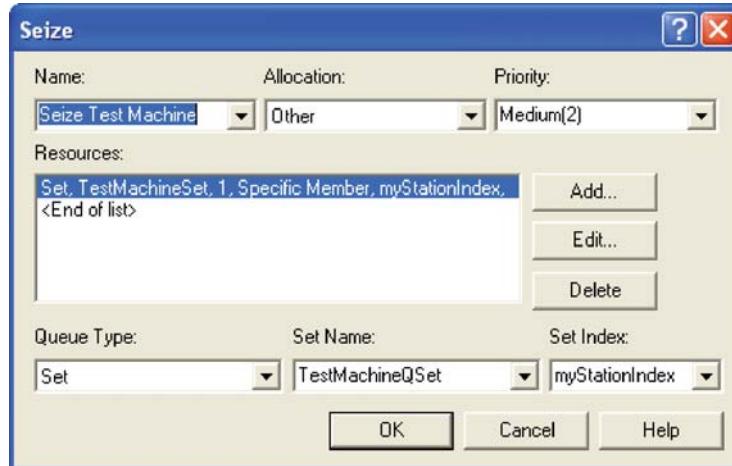


Figure 10.39 SEIZE module for generic model.

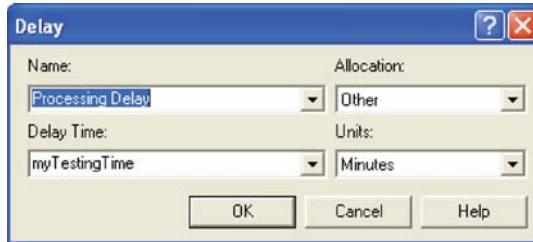
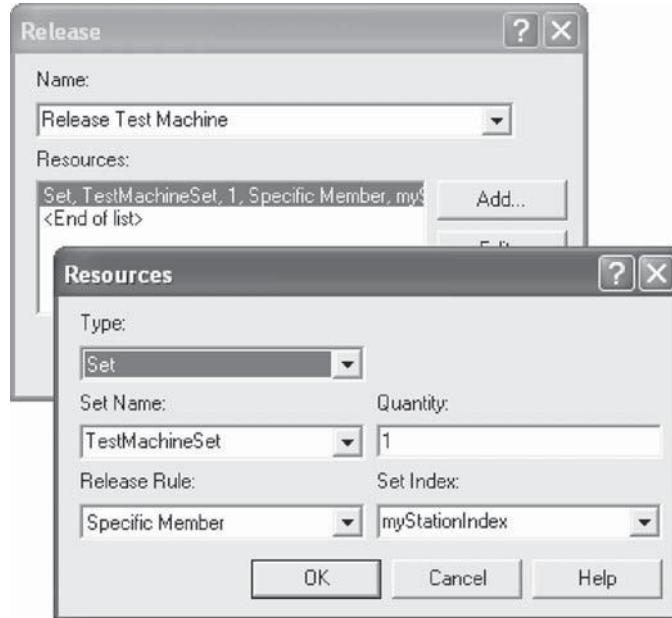


Figure 10.40 DELAY module for generic model.

Figure 10.39 illustrates the SEIZE module for the generic test and repair model. When seizing the resource, the set option has been used by selecting a specific member from the *TestMachineSet* as specified by the *myStationIndex*. Thus, if the entity had been sent to the test station 2, the *myStationIndex* would have the value 2. This value is then used as the index into the *TestMachineSet*. Now you should see why the order of the sets matters. In addition, if the entity must wait for the resource, it needs to know which queue to wait in. The SEIZE module has an option to allow the queue to be selected from a set. In this case, the *TestMachineQSet* can be accessed by using the *myStationIndex*. Since there are multiple queues being handled by this one SEIZE module, Arena™ takes away the default queue animation. You can add back the animation queues using the Animate toolbar when you develop an animation for your model.

The DELAY module for the generic station is shown in Figure 10.40. It is simple to make generic because the SEQUENCE module is being used to assign to the *myTestingTime* attribute when the entity is routed to the station.

The RELEASE module for the generic station is shown in Figure 10.41. In this case, the set option has been used to specify which resource to release. The *myStationIndex* is used to index into the *TestMachineSet* to release a specific member. The ROUTE module is just like the ROUTE module used in the original test and repair model. It routes the part using the sequence option.



**Figure 10.41** RELEASE module for generic model.

If you run this model, found in the file *GenericRepairShop.doe*, the results will be exactly as seen in Chapter 8. Now, you might be asking yourself: “This generic stuff seems like a lot of work. Is it worth it?” Well, in this case 4, flowchart modules were saved (5 in the generic versus 9 in the original model). Plus, three additional sets were added in the generic model. This does not seem like much of a savings. Now, imagine a realistically sized model with many more test stations, for example, 10 or more. In the original model, to add more testing stations, you could cut and paste the STATION, PROCESS, ROUTE combination as many times as needed, updating the modules accordingly. In the generic model, all that is needed is to add more members to the sets. You would not need to change the flowchart modules. All the changes that would need to take place to run a bigger model would be in data modules (e.g., sets and sequences). This can be a big advantage when experimenting and running different model configurations. The generic model will be much more scalable.

Until you feel comfortable with generic modeling, I recommend the following. First, you should develop the model without the generic modeling. Once the model has been tested and is working as you intended, then look for opportunities to make the model generic. Consider making it generic if you think you can take advantage of the generic components during experimentation and if you think reuse of the model is an important aspect of the modeling effort. If the model is only going to be used once to answer a specific question, then you probably do not need to consider generic modeling. Even in the case of a one-time model, if the model is very large, replacing parts of it with generic components can be a real benefit when working with the model. In this simple example, the generic modeling was easy because of the structure of the model. This will not always be the case. Thus, generic modeling will often require more thinking and adjustments than illustrated in this example; however, I have found it very useful in my modeling.

### 10.6.3 Picking up and Dropping Off Entities

In many situations, entities must be synchronized so that they flow together within the system. Chapter 5 presented how the BATCH, SEPARATE, and MATCH modules can be used to handle these types of situations. Recall that the BATCH module allows entities to wait in queue until the number of desired entities enters the BATCH module. Once the batch is formed, a single representative entity leaves the BATCH module. The batch representative entity can be either permanent or temporary. In the case of a temporary entity, the SEPARATE module is used to split the representative entity into its constituent parts.

What exactly is a *representative entity*? The representative entity is an entity created to hold the entities that form the batch in an *entity group*. An entity group is simply a list of entities that have been attached to a representative entity. The BATCH module is not the only way to form entity groups. In the case of a BATCH module, a new entity is created to hold the group. The newly created entity serves as the representative entity. In contrast, the PICKUP module causes the active entity (the one passing through the PICKUP module) to add entities from a queue to its entity group, thereby becoming a representative entity.

To get entities out of the entity group, the DROPOFF module can be used. When the active entity passes through the DROPOFF module, the user can specify which entities in the group are to be removed (dropped off) from the group. Since the entities are being removed from the group, the user must also specify where to send the entities once they have been removed. Since the entities that are dropped off were represented in the model by the active entity holding their group, the user can also specify how to handle the updating/specification of the attributes of the dropped off entities.

The PICKUP and DROPOFF modules are found on the Advanced Process panel. Figures 10.42 and 10.43 illustrate the PICKUP and DROPOFF modules. In the PICKUP module, the quantity field specifies how many entities to remove from the specified queue starting at the given rank. For example, to pick up all the entities in the queue, you can use NQ(queue name) in the quantity field.

The number of entities picked up cannot be more than the number of entities in the queue or a run time error will result. Thus, it is very common to place a DECIDE module in front of the PICKUP module to check how many entities are in the queue. This is very useful when picking up one entity at a time. When an entity is picked up from the specified queue, it is added to the group of the entity executing the PICKUP module. The entities that are picked up are added to the end of the list representing the group.

Since entities can hold other entities within a group, you may need to get access to the entities within the group during model execution. There are a number of special-purpose



Figure 10.42 PICKUP module.

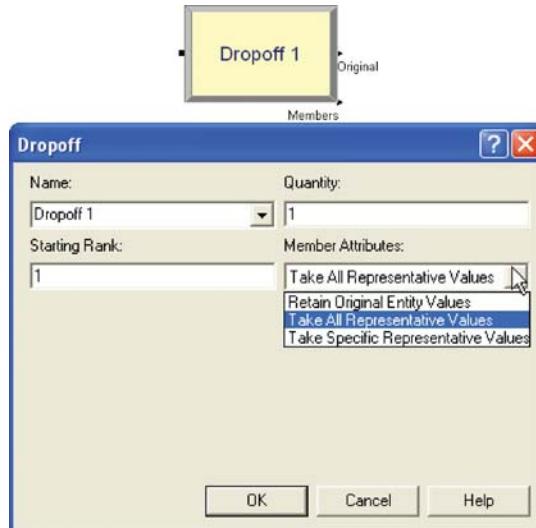


Figure 10.43 DROPOFF module.

functions/variables that facilitate working with entity groups. From the help system, information on the following functions is available:

**AG (Rank, Attribute Number)** AG returns the value of general-purpose attribute Attribute Number of the entity at the specified Rank in the active entity's group. The function NSYM may be used to translate an attribute name into the desired Attribute Number.

**ENTINGROUP (Rank [, Entity Number]**) ENTINGROUP returns the entity number (i.e., IDENT value) of the entity at the specified Rank in the group of entity representative Entity Number.

**GRPTYPE (Entity Number)** When referencing the representative of a group formed at a BATCH module, GRPTYPE returns 1 if it is a temporary group and 2 if it is a permanent group. If there is no group, then 0 will be returned.

**ISG (Rank)** This function returns the special-purpose jobstep (Entity.Jobstep, or IS) attribute value of the entity at the specified Rank of the active entity's group.

**MG (Rank)** This function returns the special-purpose station (Entity.Station, or M) attribute value of the entity at the specified Rank of the active entity's group.

**NSG (Rank)** This function returns the special-purpose sequence (Entity.Sequence, or NS) attribute value of the entity at the specified Rank of the active entity's group.

**NG (Entity Number)** NG returns the number of entities in the group of representative Entity Number. If Entity Number is defaulted, NG returns the size of the active entity's group.

**SAG (Attribute Number)** SAG adds the values of the specified Attribute Number of all members of the active entity's group. The data type of the specified attribute must be numeric. The function NSYM may be used to translate an attribute name into the desired Attribute Number.

Notice that many of the functions require the rank of the entity in the group. That is, the location in the list holding the group of entities. For example, AG(1, NSYM(*myArrivalTime*)) returns the value of the attribute named *myArrivalTime* for the first entity in the group. Often a SEARCH module is used to search the group to find the index of the required entity.

The DROPOFF module is the counterpart to the PICKUP module. The DROPOFF module removes the quantity of entities starting at the provided rank from the entity's group. The original entity exits the exit point labeled *Original*. Any members dropped off exit the module through the point labeled *Members*. If there are no members in the group, then the original simply exits. Note that similar to the case of SEPARATE, you can specify how the entities being dropped will handle their attributes. The option to *Take Specific Representative Values* causes an additional dialog box to become available that allows the user to specify which attributes the entities will get from the representative entity. The functions discussed earlier can be used on the members of the group to find the rank of specific members to be dropped off. Thus, entities can be dropped off singly or in the quantity specified. The representative entity will continue to hold the other members in the group. The special variable *NG* is very useful in determining how many entities are in the group. Again, the SEARCH module can be useful to search the group to find the rank of the entities that need to be dropped off.

To illustrate the PICKUP and DROPOFF modules, the modeling of a simple shuttle bus system will be examined. In this system, there are two bus stops and one bus. Passengers arrive to bus stop 1 according to a Poisson arrival process with a mean rate of 6 per hour. Passengers arrive to bus stop 2 according to a Poisson arrival process with a mean rate of 10 per hour. Passengers arriving at bus stop 1 desire to go to bus stop 2, and passengers arriving to bus stop 2 desire to go to bus stop 1. The shuttle bus has 10 seats. When the bus arrives to a stop, it first drops off all passengers for the stop. The passenger disembarking time is distributed according to a lognormal distribution with a mean of 8 seconds and a standard deviation of 2 seconds per passenger. After all the passengers have disembarked, the passengers waiting at the stop begin the boarding process. The per passenger boarding time (getting on the bus and sitting down) is distributed according to a lognormal distribution with a mean of 12 seconds and a standard deviation of 3 seconds. Any passengers who cannot get on the bus remain at the stop until the next bus arrival. After the bus has completed the unloading and loading of the passengers, it travels to the next bus stop. The travel time between bus stops is distributed according to a triangular distribution with parameters (16, 20, 24) in minutes. This system should be simulated for 8 hours of operation to estimate the average waiting time of the passengers, the average system time of the passengers, and the average number of passengers left at the stop because the bus was full.

Upon first analysis of this situation, it might appear that the way to model this situation is with a transporter to model the bus; however, the mechanism for dispatching transporters does not facilitate the modeling of a bus route. In standard transporter modeling, the transporter is requested by an entity. As you know, most bus systems do not have the passengers request a bus pick up. Instead, the bus driver drives the bus along its route stopping at the bus stops to pick up and drop-off passengers. The trick to modeling this system is to model the bus as an entity. The bus entity will execute the PICKUP and DROPOFF modules. The passengers will also be modeled as entities. This will allow the tabulation of the system time for the passengers. In addition, since the passengers are entities, a HOLD module with an *infinite hold* option can be used to model the bus stops. That is really all there is to it! Actually, since the logic that occurs at each bus stop is the same, the bus

stops can also be modeled using generic stations. Let us start with the modeling of the passengers.

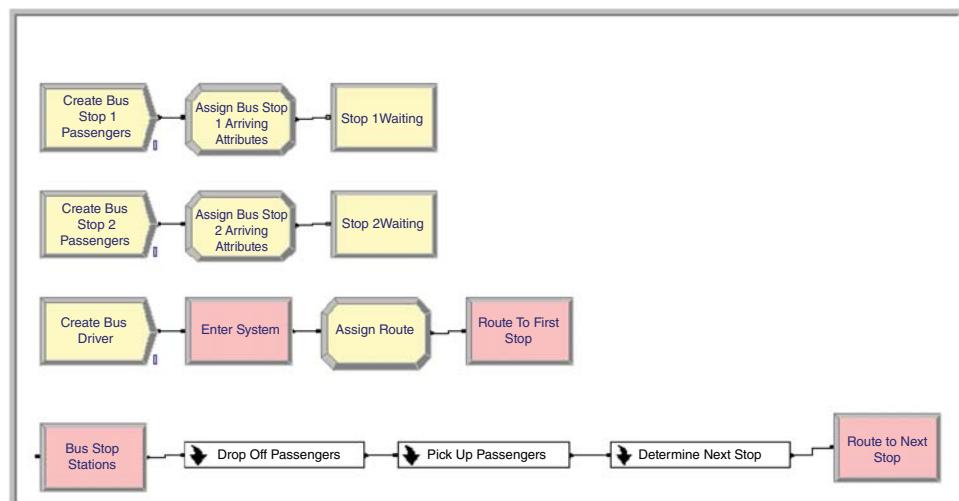
Figure 10.44 presents an overview of the bus system model. The entire model can be found in the file *ch10BusSystem.doe* associated with this chapter. In the figure, the top six modules represent the arrival of the passengers. The pseudo-code is straightforward:

- CREATE passenger with time between arrivals exponential with mean 10 minutes
- ASSIGN arrival time
- HOLD until removed.

The next four modules in Figure 10.44 represent the creation of the bus. The bus entity is created at time 0 and assigned the sequence that represents the bus route. A sequence module was used called *BusRoute*. It has two job steps representing the bus stops, *BusStation1* and *BusStation2*. Then the bus is routed using a ROUTE module according to the By Sequence option. Since these are all modules that have been previously covered, all the details will not be presented; however, the completed dialogs can be found in the accompanying model file. Now let us discuss the details of the bus station modeling.

As previously mentioned, a generic station can be used to represent each bus stop. Figure 10.45 shows the generic station for the bus stops. The station type has been defined as “Set” and the stations representing the bus stops *BusStation1* and *BusStation2* have been added to the station set. The key to the generic modeling will be the attribute, *myBusStop*, which will hold the index of the current station. In Figure 10.44, three submodels have been used to represent the drop-off logic, the pickup logic, and the logic to decide on the next bus stop to visit.

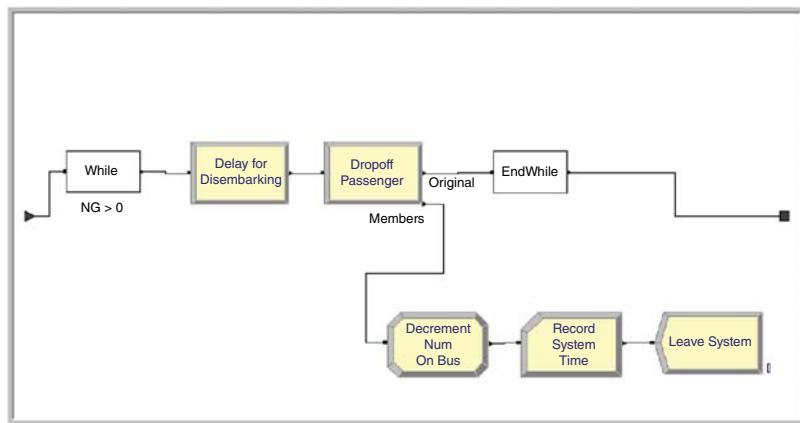
The logic for dropping off passengers is shown in Figure 10.46. This is a perfect place to utilize the WHILE-ENDWHILE blocks. Since the variable NG represents the number of entities in the active entity’s group, it can be used in the logic test to decide whether to continue to drop-off entities. This works especially well in this case since all the entities



**Figure 10.44** Bus system overview.



**Figure 10.45** Generic station for bus stops.

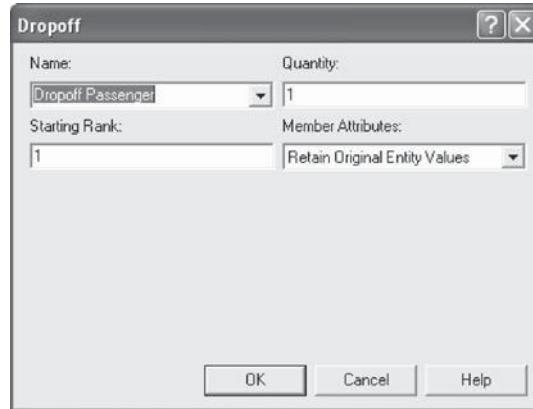


**Figure 10.46** Drop-off passengers submodel.

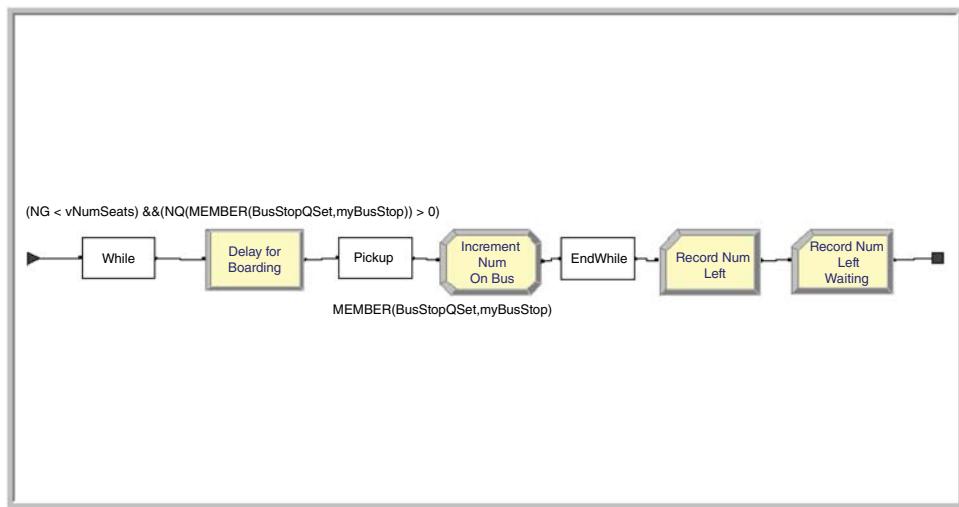
in the group will be dropped off. If the active entity (bus) has picked up any entities (passengers), then NG will be greater than zero and the statements in the while construct will be executed. The bus experiences a DELAY to represent the disembarking time of the passenger and then the DROPOFF module, see Figure 10.47, is executed. Notice that in this case, since the system time of the passengers must be captured, the *Retain Original Entity Values* option can be used.

The details of the rest of the modules can be found in the accompanying model file. In the case of a bus system with more stops, logic would be needed to search the group for passengers that want to get off at the current stop. The reader will be asked to explore this idea in the exercises.

The logic to pick up passengers is very similar to the logic for dropping off the passengers (Figure 10.48). In this case, a WHILE-ENDWHILE construct will again be used, as shown in Figure 10.42; however, there are a couple of issues that need to be handled carefully in this submodel.



**Figure 10.47** Dropping off the passengers.



**Figure 10.48** Pick up passengers submodel.

In the drop-off passenger submodel, the bus stop queues did not need to be accessed; however, when picking up the passengers, the passengers must be removed from the appropriate bus stop waiting queue. Let us take a careful look at the logical test condition for the WHILE block:

$$(NG < vNumSeats) \&\& (NQ(MEMBER(BusStopQSet, myBusStop)) > 0)$$

A variable *vNumSeats* has been defined which represents the capacity of the bus. The special-purpose group variable, *NG*, is compared to the capacity, as long as *NG* is less than the capacity the next passenger can be picked up from the bus stop queue. What about the second part of the test? Recall that it is a logical error to try to pick up an entity from an empty queue. Thus, the statement *NQ(queue name) > 0* tests to see if the queue holds any entities. Since generic modeling is being used here, the implementation must determine which queue to check based on the bus stop that the bus is currently serving. A Queue Set



**Figure 10.49** The PICKUP block.

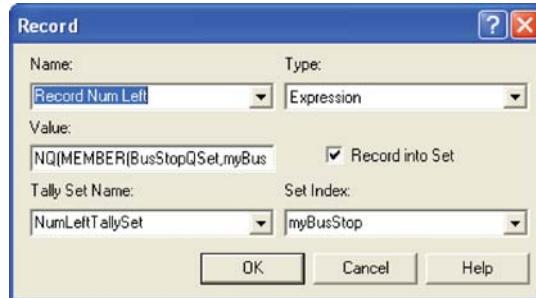
called *BusStopQSet* has been defined to hold the bus stop waiting queues. The queues have been added to this set in the same order as the bus stop stations were added to the bus stop station set. This ensures that the index into the set will return the queue for the appropriate station.

The function MEMBER(set name, index) returns the member of the set for the supplied index. Thus, in the case of MEMBER(*BusStopQSet,myBusStop*), the queue for the current bus stop will be returned. In this case, both *NG* and *NQ* need to be checked so that passengers are picked up as long as there are waiting passengers and the bus has not reached its capacity. If a passenger can be picked up, the bus delays for the boarding time of the passenger and then the PICKUP block is used to pick up the passenger from the appropriate queue. The PICKUP block, shown in Figure 10.49, is essentially the same as the PICKUP module from the Advanced Process panel, however, with one important exception. The PICKUP block allows an expression to be supplied in the Queue ID dialog field that evaluates to a queue. The PICKUP module from the Advance Process panel does not allow this functionality, which is critical for the generic modeling.

In this situation, the MEMBER(*BusStopQSet,myBusStop*) is used to return the proper queue in order to pick up the passengers. Each time the PICKUP block is executed, it removes the first passenger in the queue.

After the passengers are picked up at the stop, the logic to record the number left waiting is executed. In this case, a snap shot view of the queue at this particular moment in time is required. This is an observation-based statistic on the number in queue. Figure 10.50 shows the appropriate RECORD module. In this case, a tally set can be used so that the statistics can be collected by bus stop. The expression to be observed is NQ(MEMBER(*BusStopQSet,myBusStop*))

Running the model for 10 replications of 8 hours yields the results shown in Figure 10.51. Notice that the waiting time for the passengers is a little over 20 minutes for the two queues. In addition, the results indicate that on an average, there is less than one passenger left waiting because the bus is full. The reader will be asked to explore this model further in the exercises.



**Figure 10.50** Recording the number of passengers not picked up.

Waiting Time		
	Average	Half Width
Stop1Waiting.Queue	20.7487	1.24
Stop2Waiting.Queue	23.2777	2.90
<b>Other</b>		
Number Waiting		
	Average	Half Width
Stop1Waiting.Queue	2.0739	0.21
Stop2Waiting.Queue	3.9714	0.80
<b>User Specified</b>		
<b>Tally</b>		
Expression		
	Average	Half Width
Record Num Left Waiting	0.2567	0.29
Interval		
Record System Time		
	Average	Half Width
Record System Time	42.9741	1.65
<b>Usage</b>		
None		
	Average	Half Width
NumLeftQ1	0.00	0.00
NumLeftQ2	0.5182	0.58

**Figure 10.51** Bus system results.

Based on this example, modeling with the PICKUP and DROPOFF modules is not too difficult. The additional functions, for example, AG(), which have been not illustrated, are especially useful when implementing more complicated decision logic that can select specific entities to be picked up or dropped off. In addition, with the generic representation for the bus system, it should be clear that adding additional bus stops can be readily accomplished.

The coverage of the major modeling constructs within Arena™ has been completed. The next section closes out this chapter with some additional insights into how to use some of the programming aspects of Arena™ within your modeling projects.

## 10.7 PROGRAMMING CONCEPTS WITHIN ARENA

Within Arena™, programming support comes in two forms: laying down flowchart modules and computer language integration (e.g., VBA and C). This section presents some common programming issues that are helpful to understand when trying to get the most out of your models. First, we will examine how Arena™ stores the output from a simulation run. Then, the discussion involving input and output started in Chapter 5 will be continued. Finally, the use of VBA within the Arena™ environment will be introduced.

### 10.7.1 Using the Generated Access File

Each time the simulation experiment is executed, Arena™ writes the statistical information collected during the run to a Microsoft Access database that has the same name as the model file. This database is what is used by the Crystal Reports report generator. You can also access this database and extract the statistical information yourself. This is useful if you need to post process the statistical information in a statistical package.

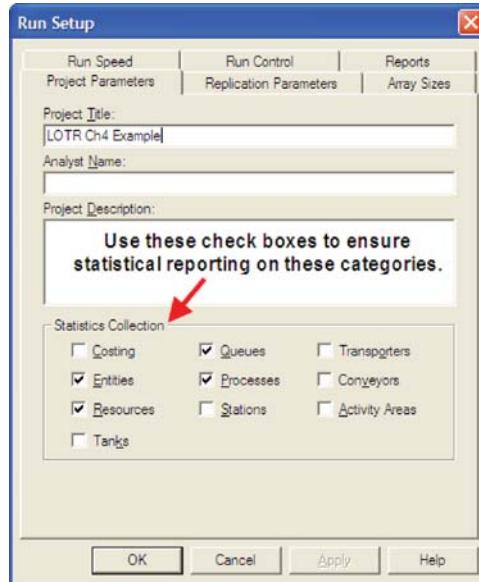
This section demonstrates how to extract information from this database. The intention is to make you aware of this database and to give you enough information so that you can use the database for some simple data extraction. There is detailed information concerning the Reports Database within the Help system. You should search under *The Report Database*. We will be reusing the LOTR example model from Chapter 7 within this section to illustrate these concepts.

When you create and run a model, for example, *yourmodel.doe*, running the model will create a corresponding Microsoft Access file called *yourmodel.mdb*. Each time the model is run, the statistical information from the simulation is written to the database. If you change the input parameters to your model (without doing anything else), and then rerun the model, that model's results are written to the database. Any previous data will be overwritten. There are two basic ways to save information from multiple simulation runs. The first is to simply rename the database file to a different name before rerunning the simulation. In this case, a new database file for the model will be created when it is run. This approach creates a new database for each simulation execution.

The second approach is to use the Projects Parameters panel within the Run > Setup menu dialog as illustrated in Figure 10.52. The Project Title field identifies a particular project. This field is used by the database to identify a given project's results. If this name is not changed when the simulation is rerun, then the results are overwritten for the named project. Thus, by changing this name before each experiment, the database will contain the information for each experiment accessible by this project name. Now let us take a look at the type of information that is stored within the database.

The Reports database consists of a number of tables and queries. A table is a set of records with each row in the table having a unique identifier called its primary key and a set of fields (columns) that hold information about the current record. A query is a database instruction for extracting particular information from a table or a set of related tables.

The help system describes in detail the structure of the database. For accessing statistical information, there are a few key tables.



**Figure 10.52** Run Setup statistical collection.

**Definition** Holds the name of the statistical item from within the model, its type (e.g., DSTAT, TALLY, and FREQ) and information about its collection and reporting.

**Statistic** The summary within replication results for a given statistic on a given replication.

**Count** The ending value for a RECORD module designated as Count for a given replication.

**Frequency** The ending values for tabulated frequency statistics by replication.

**Output** The value of a defined OUTPUT statistic for each replication.

The Definition table with the Statistic table can be used to access the replication information. Figure 10.53 illustrates the fields within the Definition table. The important fields in the Definition table are ID, Name, and *DefinitionTypeID*. The ID is the primary key of the table, Name represents the name to appear on the report that was assigned within the model for the statistic, and *DefinitionTypeID* indicates the type of statistical element. For the purposes of this section, the statistic types of interest are DSTAT (time based), TALLY (observation based), COUNTER (RECORD with count option), OUTPUT (captured at end of replication), and FREQUENCY (tabulates percentage of time spent in defined categories).

The Statistic table holds the statistical values for the within-replication statistics as shown in Figure 10.54. Notice that the Statistic table has a field called *DefinitionID*. This field is a database foreign key to the Definition table. With this field, you can relate the two tables together and get the name and type of statistic.

To explore these tables, you will work with the database called *CH10DB-Stat-Example.mdb*. This file was produced by running the *LOTRCh7Example.doe* with 56 replications and renaming the created database file. If you have Microsoft Access, then open the database and then open the table called Definition. Select the desired row, for example, row 8, and

This screenshot shows the 'Definition : Table' dialog box. It contains a table with columns for Field Name, Data Type, and Description. The table includes fields such as ID, ReportID, Name, Format, ReportLineDefinitionID, ArgumentIndex, Expression, Type, RunOutputID, OutputFileID, SourceDataTypeID, SourceCategoryID, SourceProcessID, DefinitionTypeID, and Limit. The 'Description' column provides detailed explanations for each field.

Field Name	Data Type	Description
ID	AutoNumber	
ReportID	Number	Report that this line belongs to
Name	Text	Name to be used to identify this report line
Format	Text	Format specifier in C or FORTRAN format, refer to documentation for limitations in database processing
ReportLineDefinitionID	Number	
ArgumentIndex	Number	Order of the argument in the parameter list
Expression	Text	Expression used to develop the value
Type	Text	Data type for report line, typically SMINT, SMREAL, STR
RunOutputID	Number	The run that is associated with this row
OutputFileID	Number	
SourceDataTypeID	Number	Data type (like VACost or NVATime)
SourceCategoryID	Number	Generic type (like Entity, Resource, or Queue), or that it is a particular template and module (like "Basic-Create" or "Advanced-Process")
SourceProcessID	Number	Identify the module
DefinitionTypeID	Number	Type of statistical element represented i.e.: DSTAT, CSTAT, TALLY ..... foreign key to DefinitionTypes
Limit	Number	

Figure 10.53 Definition table field design.

This screenshot shows the 'Statistic : Table' dialog box. It contains a table with columns for Field Name, Data Type, and Description. The table includes fields such as ID, ReplicationID, DefinitionID, MinObs, MaxObs, AvgObs, HalfWidth, LastValue, NumObs, and StdDev. The 'Description' column provides detailed explanations for each field.

Field Name	Data Type	Description
ID	AutoNumber	
ReplicationID	Number	
DefinitionID	Number	
MinObs	Number	The minimum observation value within this scenario/replication.
MaxObs	Number	The maximum observation value within this scenario/replication.
AvgObs	Number	The average observation value for this scenario/replication.
HalfWidth	Number	The .95 half width for this scenario/replication.
LastValue	Number	The final variable value for this scenario/replication.
NumObs	Number	The number of observations in this stat
StdDev	Number	The standard deviation of observations

Figure 10.54 Statistic table field design.

use Insert > Subdatasheet to get the Insert Subdatasheet dialog as shown in Figure 10.55. Then, select the Statistic table and press the OK button as shown in the figure.

Figure 10.56 shows the result of inserting the linked data sheet and selecting the “+” sign for the statistic named “Record Make and Inspect Time.” You should right-click on the *ReplicationID* field and sort the subdatasheet by increasing replication number. From this table, you can readily assess the replication statistics. For example, the *AvgObs* column refers to the ending average across all the observations recorded during each replication. From this, you can easily cut and paste any required information into a spreadsheet or a statistical package such as R. For those familiar with writing queries, this same information can be extracted by writing the query as shown in Figure 10.57.

To summarize the statistics across the replications, you can write a group by query as shown in Figure 10.58.

In the original LOTR Ring Maker, Inc. model, an OUTPUT statistic was defined to collect the time that the simulation ended. The OUTPUT statistic values collected at the end of each replication are saved in the Output table as shown in Figure 10.59.

To see the collected values for a specific defined statistic, you can again use the Definition table. Open up the Definition table and scroll down to the row 48 which is the “TimeToMakeOrderStat” statistic. Select the row and using the Insert > Subdatasheet dialog, insert the Output statistic table as the subsheet. You should see the subsheet as shown in Figure 10.60. These same basic techniques can be used to examine information on the COUNTERS and FREQUENCY statistics that have been defined in the model. If you name each run with a different Project Name, then you can do queries across projects. Thus, after you have completed your simulation analysis, you do not need to rerun your model; you can simply access the statistics that were collected from within the Reports database. If you are

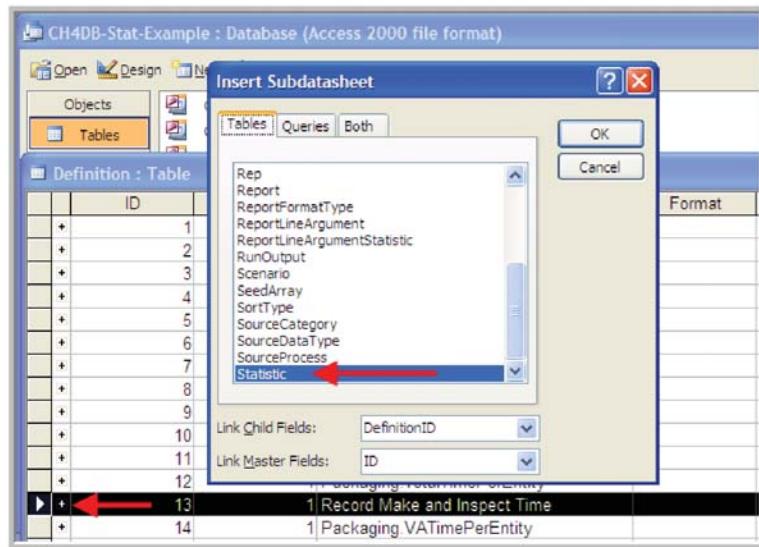


Figure 10.55 Accessing statistics information.

ID	ReportID	Name	Format	ReportLineDefin	ArgumentIndex
12		1 Packaging TotalTimePerEntity		0	0
13		1 Record Make and Inspect Time		0	0
13	1	11.4505300186	705 241399798	363 804363517	2E+20 705 241399798
56	2	18.2185229538	700 783930168	346 058804617	2E+20 700 783930168
99	3	15.5303417880	496 832758865	259 099768822	2E+20 496 832758865
142	4	18.1116237371	659 372679112	328 811820096	2E+20 659 372679112
185	5	11.6530644408	879 533299880	449 363748219	2E+20 879 533299880
228	6	12.5886395300	708 877835415	365 714157645	2E+20 708 877836415
271	7	16.4424930053	664 036713358	339 121842004	2E+20 664 036713358
314	8	19.0659549956	614 273960189	316 357093763	2E+20 614 273960189
357	9	12.8852738538	941 455156957	486 854846717	2E+20 941 455156957
400	10	15.9273709035	806 236078717	402 755366380	2E+20 806 236078717

Figure 10.56 Replication statistics for make and inspect time.

an experienced Microsoft Access user, you can then form custom queries, reports, charts, etc. for your simulation results.

### 10.7.2 Working with Files, Excel, and Access

As discussed in Chapter 5, Arena™ allows the user to directly read from or write to files within a model during a simulation run by using the READWRITE module located on the Advanced Process panel. Using this module, the user can read from the keyboard, read from a file, write to the screen, or write to a file. When reading from or writing to a file, the user must define an Arena™ File Name and optionally a file format. The Arena™ File Name is the internal name (within the model) for the external file defined by the operating system. The internal file name is defined with the FILE data module. Using the FILE data module, the user can specify the name and path to the external file. The file format can be specified either in the FILE data module or in the READWRITE module to override

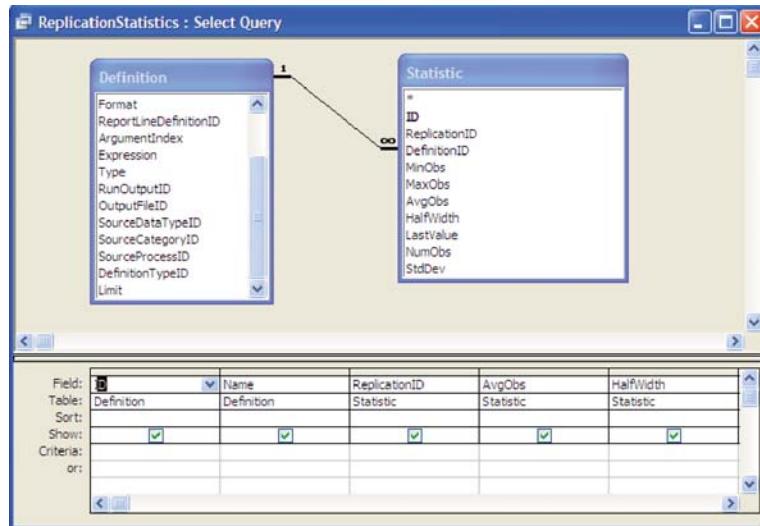


Figure 10.57 Query to access replication statistics.

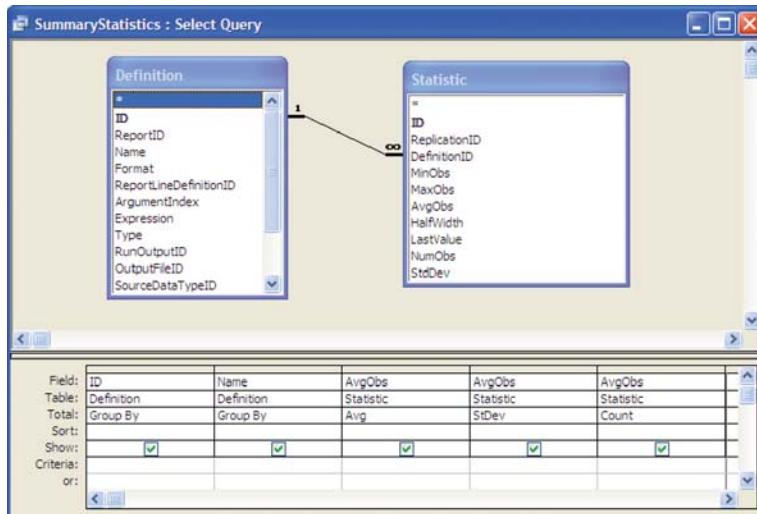


Figure 10.58 Query to summarize across replications.

the specification, given in the FILE data module. The format can be free format, a valid C or FORTRAN format, WKS for Lotus spreadsheets, Microsoft Excel, Microsoft Access, and ActiveX Data Objects Access types. In order for the READWRITE module to operate, an entity must pass through the module. Thus, as demonstrated in Chapter 5, it is typical to create a logic entity at the appropriate time of the simulation to read from or write to a file. This section presents examples of the use of the READWRITE module. Then, the pharmacy model is extended to read parameters from a database and write statistical output to a database.

Field Name	Data Type	Description
ID	AutoNumber	
DefinitionID	Number	Base definition for this counter
ReplicationID	Number	The replication number for this count data.
Value	Number	The current output value for this scenario/replication.

Figure 10.59 Output table field design.

ID	ReplicationID	Value
47	1	Make Ring Process Accum
48	1	TimeToMakeOrderStat
4	1	713.743803986
33	2	708.325141218
62	3	505.975704073
91	4	666.442097128
120	5	885.894393495
149	6	716.014367741
178	7	680.007737642
207	8	622.710463147
236	9	949.786832672
265	10	820.723639855

Figure 10.60 Expanded datasheet view for OUTPUT statistics.

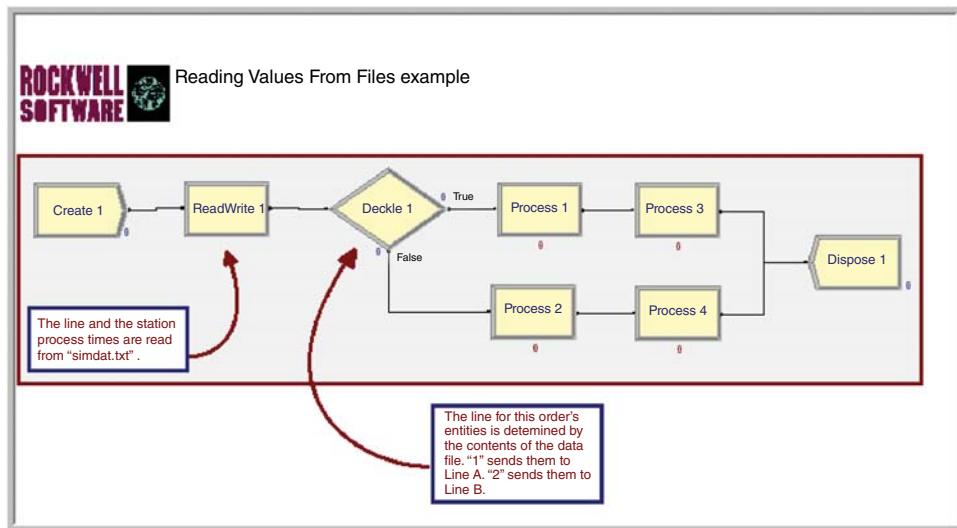
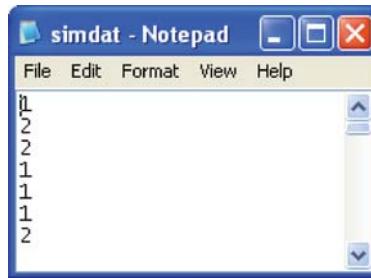


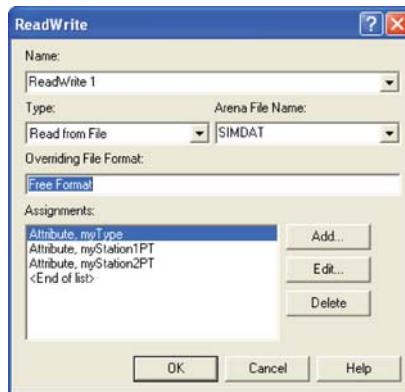
Figure 10.61 Arena™ Smarts 162 model.

**10.7.2.1 Reading from a Text File** In this example, the SMART file *Smarts162.doe* is used to show how to read from a text file (Figure 10.61). Open up the file named *Smarts162Revised.doe*.

In this model, entities arrive according to a Poisson process, the type of entity and thus the resulting path through the processing is determined via the values within the *simdat.txt* file (Figure 10.62). The first number in the file is the type 1 or 2. Then, the following two numbers are the station 1 and station 2 processing times, respectively. In Figure 10.63, the READWRITE module reads directly from the SIMDAT file using a free format. Each



**Figure 10.62** Sample processing times in simdat.txt file.

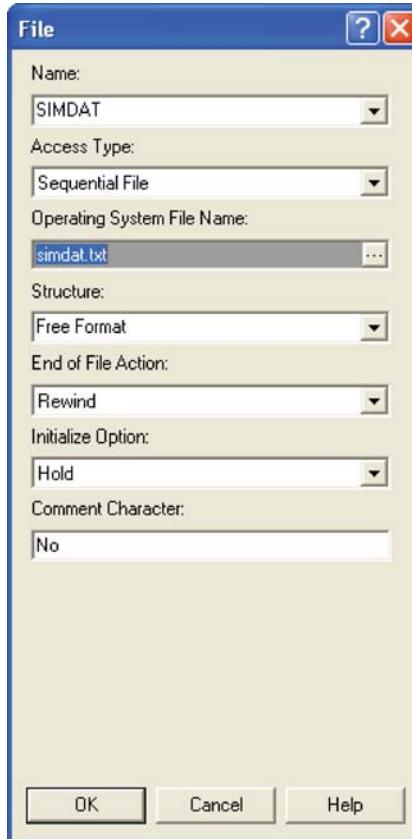


**Figure 10.63** READWRITE module.

time a read occurs, the attributes (*myType*, *myStation1PT*, *myStation2PT*) are read in. The *myType* attribute is tested in the DECIDE module, and the attributes *myStation1PT* and *myStation2PT* are used in the PROCESS modules.

The end of file action specifies what to do with the entity when the end of the file is reached. The Error option can be used if an unexpected EOF condition occurs. The Dispose option may be used to dispose of the entity, close the file or ADO recordset and stop reading from the file (Figure 10.64). The Rewind option may be used so that every time you reach an EOF, you start reading the file or recordset again from Record 1. Finally, the Ignore option can be used if you expect an EOF and want to determine your own action (such as reading another file). With the Ignore option when the EOF is reached, all variables read within the READWRITE module will be set to 0. The READWRITE module can be followed with a DECIDE module to ensure that if all values are 0, the desired action is taken. The comment character is useful to embed lines within the file that are skipped. These lines can add columns headers, comments, etc. to the file for easier readability of the file. Suppose the comment character was a ";" then

```
; This would be a comment followed by a header comment
; Num Servers\ \ Arrival Rate
1 10
```



**Figure 10.64** FILE module for Smarts162Revised.doe.

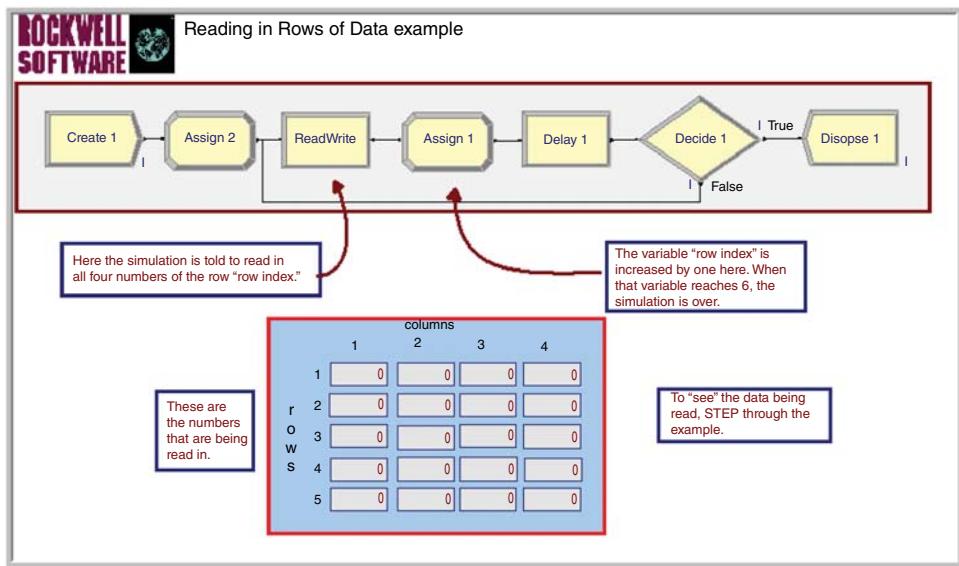
The *Initialize* option indicates what Arena™ should do at the beginning of a replication. The file can stay at the current position (Hold), be rewound to the beginning of the file (Rewind), or the file can be closed (Close).

The rest of the model is straightforward. In this case, the values from the file are read into the attributes of an entity. The values of an array can also be read in using this technique.

**10.7.2.2 Reading a Two-Dimensional Array** Smarts file *Smarts164.doe* shows how to read into a two-dimensional array (Figure 10.65). The CREATE module creates a single entity and the ASSIGN module initializes the array index. Then, iterative looping is performed using a DECIDE module as previously discussed in Chapter 5.

In this particular example, the entity delays for 10 minutes before looping to read in the next values for the array. The assignments for the READWRITE module are show in Figure 10.66. You can easily see how the use of two WHILE-ENDWHILE loops could allow for reading in the size of the array. You would first read in the number of rows and columns and then loop through each row/column combination.

**10.7.2.3 Reading from an Excel Named Range** So far the examples have focused on text files, and for simple models, these will often suffice. For more user friendly input of



**Figure 10.65** Smarts164.doe reading in a 2D array.

Assignments		
	Type	Other
1	Other	DupNum(Row Index,1)
2	Other	DupNum(Row Index, 2)
3	Other	DupNum(Row Index, 3)
4	Other	DupNum(Row Index, 4)

Double-click here to add a new row.

**Figure 10.66** READWRITE assignments module using arrays.

the data to files, you can read from Excel files. Smarts file 185 shows how to read in values from an Excel named range (Figure 10.67).

In order to read or write to an Excel named range, the named range must already be defined within the spreadsheet. Open up the Excel file *Smarts185.xls*. Select cells C5:E6 and look in the upper-left-hand corner of the Excel input bar. You will see the named range. By selecting any cells within Excel, you can name them by typing the name into the named range box as indicated in Figure 10.68. You can organize your spreadsheet in any way that facilitates the understanding of your data input requirements and then name the cells required for input. The named ranges are then accessible through the FILE module.

In this example, the processing times are distributed according to a triangular distribution. The named spreadsheet cells hold the parameters of the triangular distribution (min, mode, max) for each of the two PROCESS modules. An EXPRESSION module was used to define expressions with the variables to be read in indicating the parameter values. The order quantities are how much the customer has ordered. The lower CREATE module creates 50 arriving customers, where the order quantity is read and then used in the processing times. In the use of named ranges, essentially the execution of each READWRITE module causes a new row to be read. Thus, as indicated in Figure 10.67, there are two back-to-back READWRITE modules in the top level create logic to read in each of the rows associated

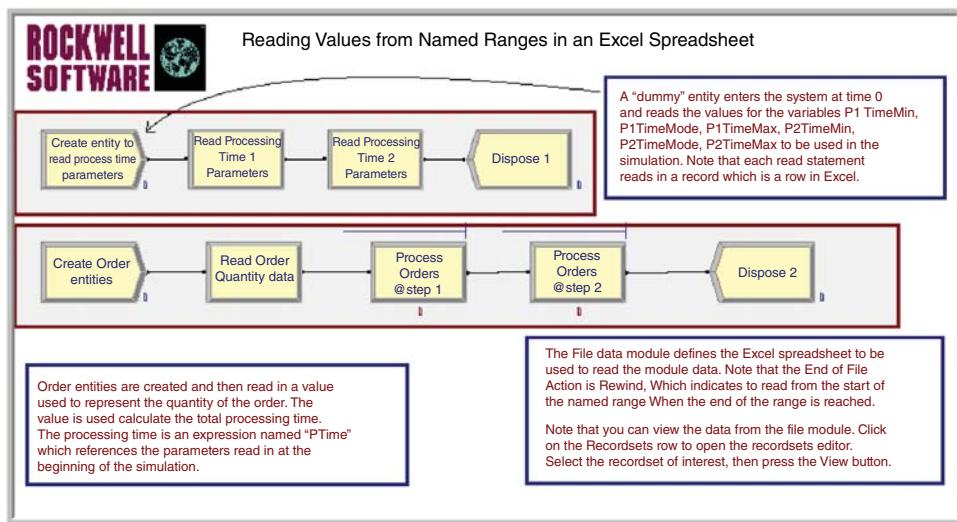


Figure 10.67 Smarts185.doe reading from an Excel named range.

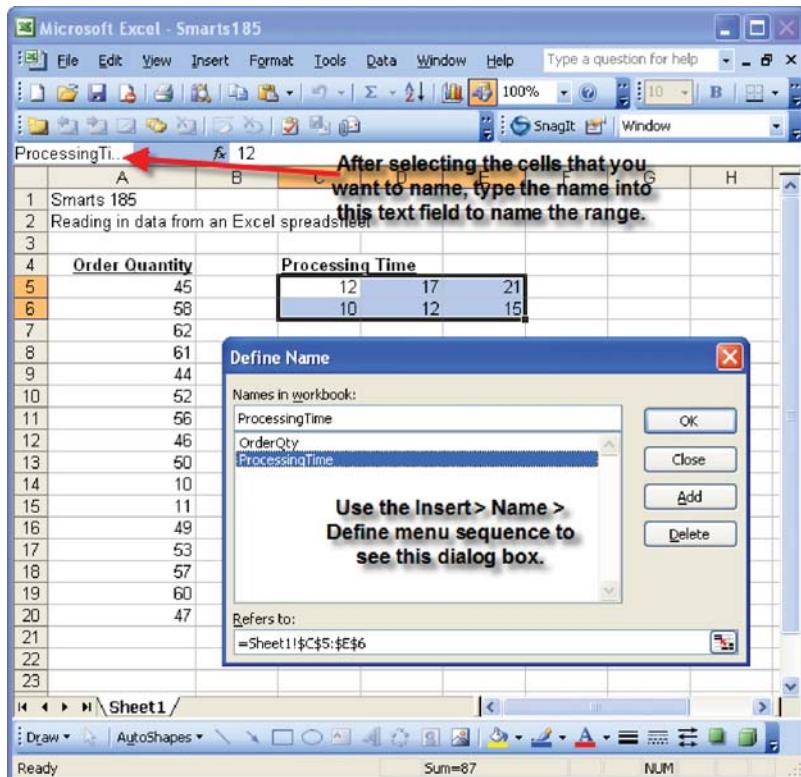


Figure 10.68 Named range in Excel.

with the processing times. In the bottom create logic, each new entity reads in a new row from the named range.

After setting up the spreadsheet and defining the named ranges within Excel, you must then define a file with the named range. This is accomplished through the use of the FILE module. Select the FILE module within *Smarts185.doe*. The FILE module is already defined for you, but the steps will be indicated here. In the spreadsheet data view, double-click to add a new row and fill in *AccessType*, Operating System File Name, End of File Action, and Initialize Option the same as in the previous row (Figure 10.69). Then, click on the define Recordsets row button.

You will see a dialog box similar to Figure 10.70. Arena™ is smart enough to connect to Excel and to populate the Named Range text box with the named ranges that you defined for your spreadsheet. You need only select your desired named ranges and click on Add/Update. This will add the named range as a record set in the Recordsets in file area. You should add both named ranges as shown in Figure 10.71. Once you select a named range you can also view the data in the range by selecting the View button. If you try this with the ProcessingTime named range, you will see a view of the data similar to that shown in Figure 10.71. As you can see, it is very simple to define a named range and connect it to Arena™.

Now, you have to indicate how to use the named range within the READWRITE module. Open up the READWRITE module, see Figure 10.72 for reading the processing time 1 parameters. After selecting read from file and the associated Arena™ file name, Arena™ will automatically recognize the file as a named range based Excel file. You can then choose the recordset that you want to read from and then the module works essentially as it did in our text file examples. You can also use these procedures to write to Excel named ranges and to Excel using active data objects (ADO). See, for example, SMARTS files 189 and 190.

Making the simulation file driven takes special planning on how to organize the model to take advantage of the data input. Using sets and arrays appropriately is often necessary when making a model file driven. Here are some ideas for using files:

	Name	Access Type	Operating System File Name	End of File Action	Initialize Option	Recordsets
1	Excel Data File	Microsoft Excel (*.xls)	Smarts185.xls	Rewind	Hold	2 rows

Double-click here to add a new row.

Figure 10.69 Datasheet view of FILE module.

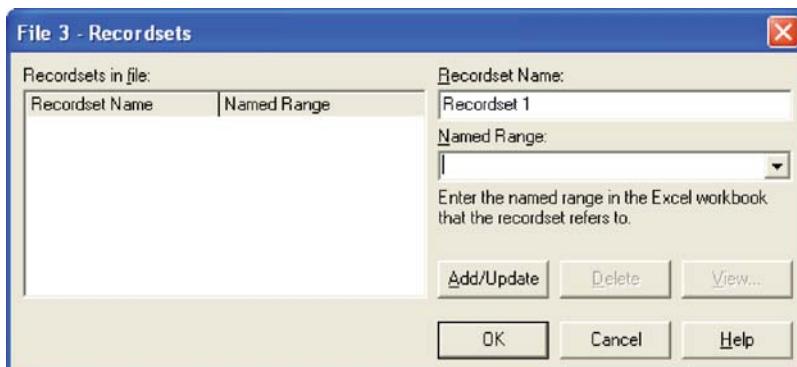
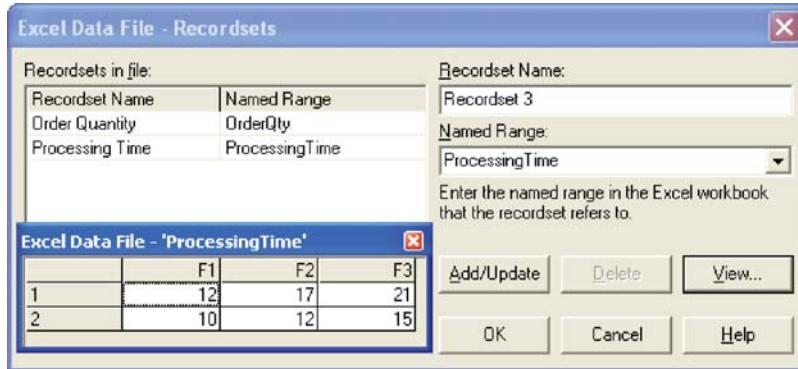
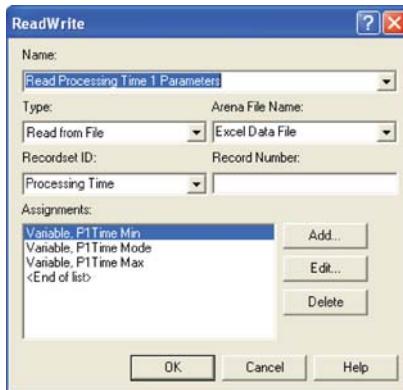


Figure 10.70 Recordset defining for FILE module.



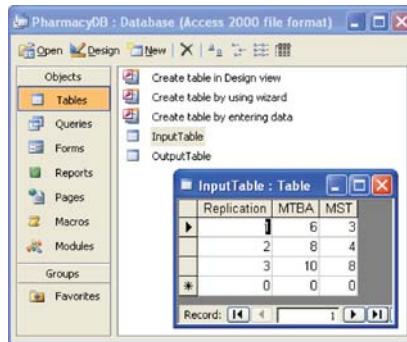
**Figure 10.71** Viewing the ProcessingTime named range within Arena™.



**Figure 10.72** READWRITE module using named range.

- Reading in the parameters of distributions.
- Reading in model parameters (e.g., reorder points and order quantities).
- Reading in specified sequences for entities to follow. Define a set of sequences and read in the index into the set for the entity to follow. You can then define different sequences based on a file.
- Reading in different expressions to be used. Define a set of expressions and read in the index into the set to select the appropriate expression.
- Reading in a number of entities to create, then create them using a SEPARATE module.
- Creating entities and assigning their attributes based on a file.
- Reading in each entity from a file to create a trace-driven simulation.

**10.7.2.4 Reading Model Variables from Microsoft Access** This example uses the pharmacy model and augments it to allow the parameters of the simulation to be read in from a database. In addition, the simulation will write out statistical data at the end of each run. This example involves the use of Microsoft Access, if you do not have Access, then just follow along with the text. In addition, the creation of the database show in Figure 10.73



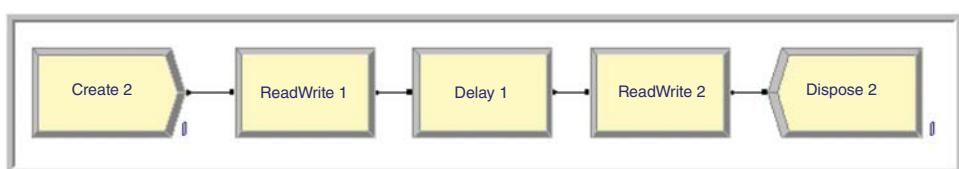
**Figure 10.73** PharmacyDB access database.

will not be discussed. The database has two tables: one to hold the inputs and one to hold the output across simulation replications. Each row in the table *InputTable* has three fields. The first field indicates the current replication, the second field indicates the mean time between arrivals for that experiment, and the last field indicates the mean service time for the experiment. This example will only have a total of three replications; however, each replication is *not* going to be the same. At the beginning of each replication, the parameters for the replication will be read in and then the replication will be executed. At the end of the replication, some of the statistics related to the simulation will be written out to the table called *OuputTable*. For simplicity, this table also only has three columns. The first column is the replication number, the second column will hold the average waiting time in the queue, and the third column will hold the half-width reported from Arena™.

Open up the *PharmacyModelRWv1.doe* file and examine the VARIABLE module. Notice that variables have been defined to hold the mean time between arrivals and the mean service time. Also, you should look at the CREATE and PROCESS modules to see how the variables are being used. The logic to read in the parameters at the beginning of each replication and to write the statistics at the end of the replication must now be implemented. The logic will be quite simple as indicated in Figure 10.74. An entity should be created at time 0, read the parameters from the database, delay for the length of the simulation, and then write out the data.

You should lay down the modules indicated in Figure 10.74. Now, the Arena™ FILE module must be defined. The process for using Access as a data source is very similar to the way Excel named ranges operate. Figure 10.75 shows the basic setup in the spreadsheet view. Opening up the recordset rows allows you to define the tables as recordsets. You should define the recordsets as indicated in the figure.

The two READWRITE modules are quite simple. In the first READWRITE module, the variables *RepNum*, *MTBA*, and *MST* are assigned values from the file (Figure 10.76).



**Figure 10.74** Read/write logic for example.



Figure 10.75 FILE module for access database.

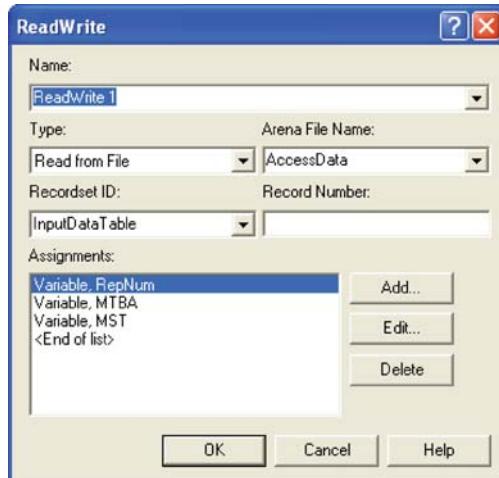
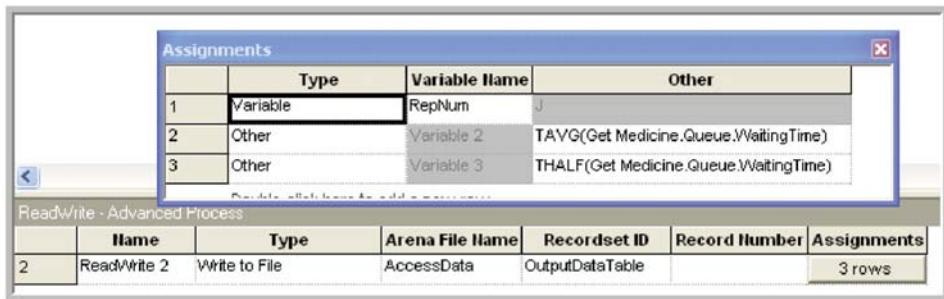


Figure 10.76 READWRITE assignments in first READWRITE module.

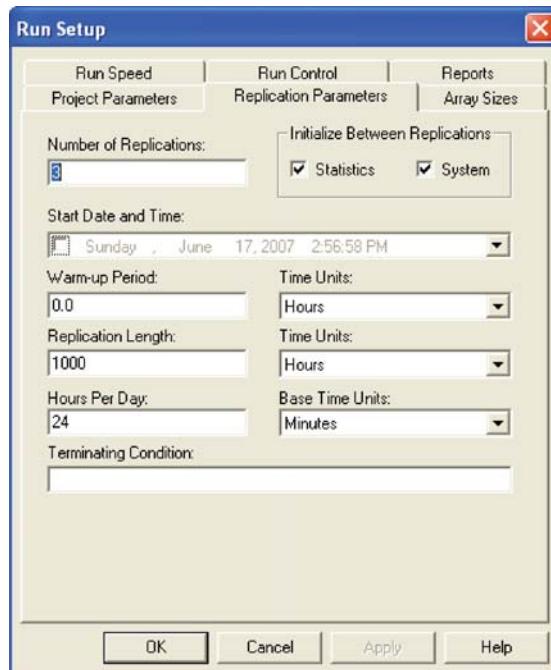
This will occur at time 0 and thereby set the parameters for the run. The second READWRITE module writes out the value of the *RepNum* and uses the Arena™ functions *TAVG()* and *THALF()* to get the values of the statistics associated with the waiting time in queue (Figure 10.77).

Now, the setup of the replications must be considered. In this model, at the beginning of each replication, the parameters will be read in. Since there are three rows in the database input table, the number of replications will be set to 3 so that all rows are read in. In addition, the run length is set to 1000 hours and the base time unit to minutes, as shown in Figure 10.78.

Only one final module is left to edit. Open up the DELAY module. The entity entering this module should delay for the length of the simulation (Figure 10.79). Arena™ has a special-purpose variable called *TFIN* which holds the length of the simulation in *base time*.



**Figure 10.77** READWRITE assignments in second READWRITE module.

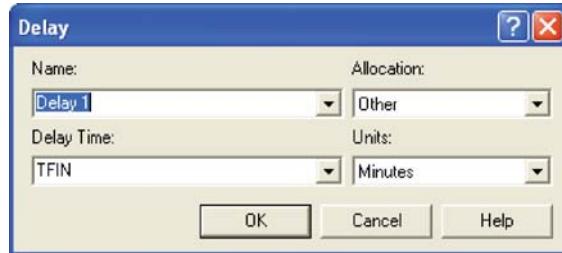


**Figure 10.78** Run Setup parameters for example.

units. Thus, the entity should delay for  $TFIN$  units. Make sure that the units match the base time units specified in the Run Setup dialog.

After the entity delays for  $TFIN$  time units, it enters the READWRITE module where it writes out the values of the statistics at that time. After running the model, you can open up the Microsoft Access database and view the *OutputTable* table (Figure 10.80). You should see the results for each of the three replications.

Suppose now you wanted to replicate each run involving the parameter settings three times. All you would need to do would be to set up your Access input table as shown in Figure 10.81 and change the number of replications to 9 in the Run Setup dialog.



**Figure 10.79** Delaying for the length of the simulation.

OutputTable : Table			
	Replication	AvgWaitingTime	HWWWaitingTime
▶	1	3.25356452762425	0.294470922917716
	2	3.81602561568035	0.373565865866371
	3	27.7529649795808	6.52722455471652
*	0	0	0

**Figure 10.80** Output within access database.

InputTable : Table			
	Replication	MTBA	MST
▶	1	6	3
	1	6	3
	1	6	3
	2	8	4
	2	8	4
	2	8	4
	3	10	8
	3	10	8
	3	10	8
▶	0	0	0

**Figure 10.81** Making repeated replications.

This same approach can be easily repeated for larger models. This allows you to specify a set of experiments, say according to an experimental design plan, execute the experiments, and easily capture the output responses to a database. Then, you can use any of your favorite statistical analysis tools to analyze your experiments. I have found that for very large experiments where I want fine-grained control over the inputs and outputs that the approach outlined here works quite nicely.

### 10.7.3 Using Visual Basic for Applications

This section discusses the relationship between Arena™ and VBA. VBA is Microsoft's macro language for applications built on top of the Windows operating system. As its name implies, VBA is based on the visual basic (VB) language. VBA is a full featured language that has all the aspects of modern computer languages include the ability to create objects with properties and methods. A full discussion of VBA is beyond the scope of this text. For an introduction to VBA for people interested in modeling, you might refer to Albright [2001].

The section assumes that the reader has some familiarity with VBA or is, at the very least, relatively computer language literate. This topic is relatively advanced and typical usage of Arena™ often does not require the user to delve into VBA. The interaction between Arena™ and VBA will be illustrated through a discussion of the VBA block and the use of Arena's user-defined function. Through VBA, Arena™ also has the ability to create models (e.g., lay down modules and fill dialogs) via Arena's VBA automation model. This advanced topic is not discussed in this text, but extensive help is available on the topic via the online help system.

To understand the connection between Arena™ and VBA, you must understand the user event-oriented nature of VBA. Do not confuse the user interaction events discussed in this section with discrete events that occur within a simulation. VB was developed as an augmentation of the BASIC computer language to facilitate the development of visual user interfaces. User interface interaction is inherently event driven. That is, the user causes various events to occur (e.g., move the mouse and clicks a button) that the user interface must handle. To build a program that interacts significantly with the user involves writing code to react to specific events. This paradigm is quite useful, and as VB became more accepted, the event-driven model of computing was expanded beyond that of directly interacting with the user. Additional nonuser interaction events can be defined and can be called at specific places within the execution of the program. The programmer can then place code to be called at those specific times in order to affect the actions of the program. Arena™ is integrated with VBA through a VBA event model.

There are a number of VBA events that are predefined within Arena's VBA interaction model. The following key VBA events will be called automatically if an event routine is defined for these events in VBA within Arena™.

**DocumentOpen, DocumentSave** These events are called when the model is opened or saved. The SIMAN object is not available, but the Arena™ object can be used. The SIMAN object will be discussed within some examples.

**RunBegin** This event is called prior to the model being checked. When the model is checked, the underlying SIMAN code is compiled and translated to executable machine code. You can place code within the RunBegin event to add or change modules with VBA automation so that the changes get complied into the model. The SIMAN object is not available, but the Arena™ object can be used. This event occurs automatically when the user invokes a simulation run. The Arena™ object will not be discussed in this text, but plenty of help is available within the help system.

**RunBeginSimulation** This event is called prior to starting the simulation (i.e., before the first replication). This is a perfect location to set data that will be used by the model during all replications. The SIMAN object is active when this event is fired and remains active until after RunEndSimulation fires. Because the simulation

is executing, changes using the Arena™ object via automation are not permissible.

**RunBeginReplication** This event is called prior to the start of each replication.

**OnClearStatistics** This event is called if the clear statistics option has been selected in run setup. This event occurs prior to each replication and at the designated warm-up time if a warm-up period has been supplied.

**RunEndReplication** This event is called at the end of each replication. It represents a perfect location for capturing replication statistical data. It is only called if the replication completes naturally with no interruption from errors or the user.

**RunEndSimulation** This event is called after all replications are completed (i.e., after the last replication). It represents a perfect place to close files and get across replication statistical information.

**RunPause, RunRestart, RunResume, RunStep, RunFastForward, RunBreak** These events occur when the user interacts with Arena's run control (e.g., pauses the simulation via the pause button on the VCR control). These events are useful for prompting the user for input.

**RunEnd** This event is called after RunEndSimulation and after the run has been ended. The SIMAN object is no longer active in this event.

**OnKeystroke, OnFileRead, OnFileWrite, OnFileClose** These events are fired by the SIMAN runtime engine if the named event occurs.

**UserFunction, UserRule** The UserFunction event is fired when the UF function is used within the model. The UserRule event is fired when the UR function is called within the model.

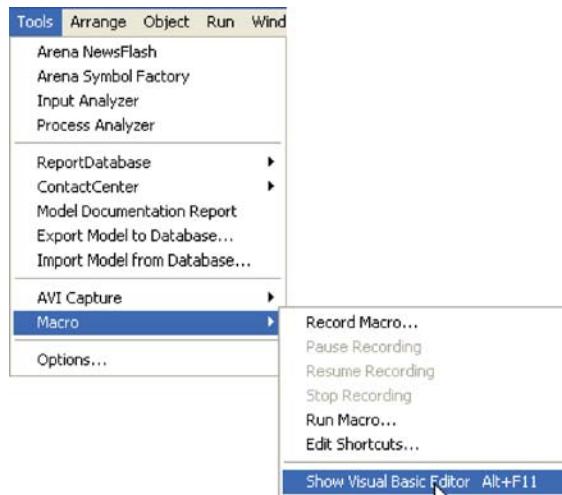
**SimanError** This event is called if SIMAN encounters an error or warning. The modeler can use this event to trap SIMAN-related errors.

A VBA event is defined, if a subroutine is written that corresponds to the VBA event naming convention within the *ThisDocument* module accessed through the VBA editor. In VBA, a file that holds code is called a module. This should not be confused with an Arena™ module.

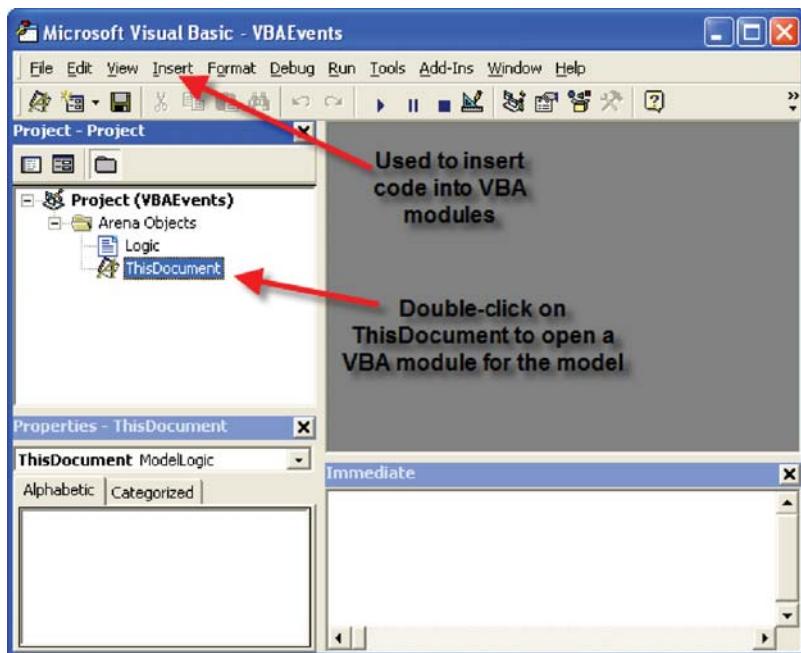
**10.7.3.1 Using VBA** Let us take a look at how to write a VBA event for responding to the *RunBegin* event. The Arena™ file, *VBAEvents.doe* that is available with this chapter, should be opened. The Arena™ model is a simple CREATE, PROCESS, DISPOSE combination to create entities and have a model to illustrate VBA. The specifics of the model are not critical to the discussion here. In order to write a subroutine to handle the *RunBegin* event, you must use the VBA Editor. Within the Arena™ environment, use the Tools > Macro > Show Visual Basic Editor menu option (as shown in Figure 10.82).

This will open the VBA Editor as shown in Figure 10.83. If you double-click on the *ThisDocument* item in the VBA projects tree as illustrated in Figure 10.83, you will open up a VBA module that is specifically associated with the current model.

A number of VBA events have already been defined for the model. Let us insert an event routine to handle the *RunBegin* event. Place your cursor on a line within the *ThisDocument* module and go to the event drop down box called (General). Within this drop down box, select Model Logic. Now go to the adjacent drop down box that lists the available VBA events. Clicking on this drop down list will reveal all the possible Arena™ VBA events that are available, as shown in Figure 10.84. The event routines that have already been



**Figure 10.82** Showing the Visual Basic Editor.



**Figure 10.83** The VBA Editor.

written are in bold. As can be seen the *OnClearStatistics* event is indicating that it has been written and this can be confirmed by looking at the code associated with the *ThisDocument* module. At the end of the list is the *RunBegin* event. Select the *RunBegin* event and an event routine will automatically be placed within the *ThisDocument* module at your current cursor location.

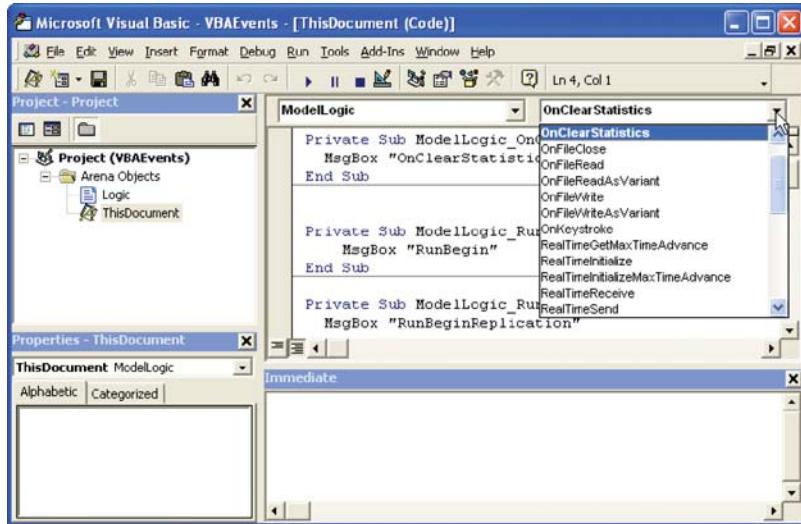


Figure 10.84 Arena's VBA events.

The event procedure has a very special naming convention so that it can be properly called when the VBA integration mechanism needs to call it during the execution of the model. The code will be very simple, it will just open up a message box that indicates that the *RunBegin* VBA event has occurred (Figure 10.85). The code to open up a message box when the event procedure fires is as follows:

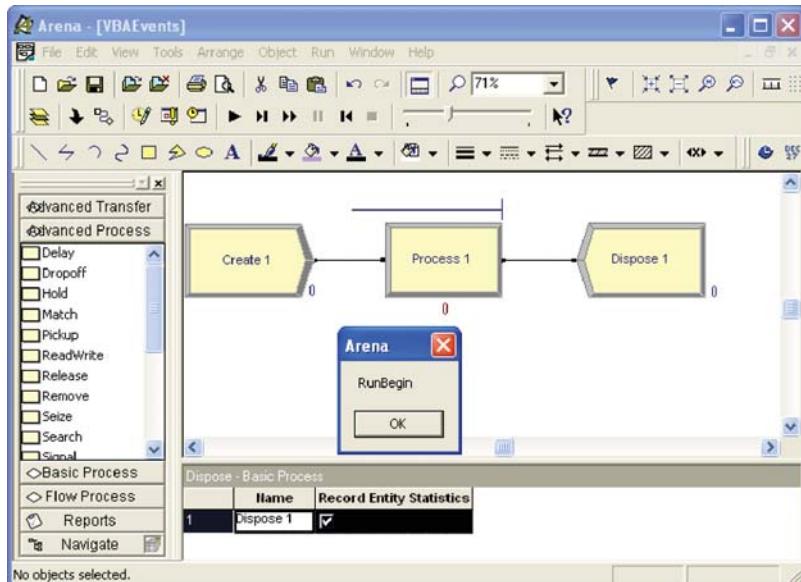


Figure 10.85 Message box for RunBegin example.

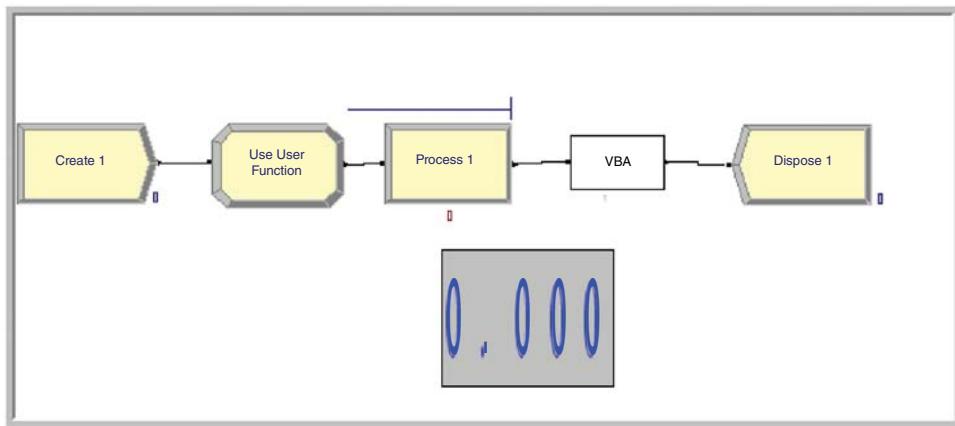
```
Private Sub ModelLogic_RunBegin()  
    MsgBox "RunBegin"  
End Sub
```

The use of the *MsgBox* function in this example is just to illustrate that the subroutine is called at the proper time. You will quite naturally want to put more useful code within your VBA event subroutines.

A number of similar VBA event subroutines have been defined with similar message boxes. Go back to the Arena™ environment and press the run button on the run control toolbar. As the model executes, a series of message boxes will open up. After each message box appears, press okay and continue through the run. As you can see, the code in the VBA event routines is executed when Arena™ fires the corresponding event. The use of the message box here is a bit annoying, but clearly indicates where in the sequence of the run that the event occurs.

The Arena™ Smart files are a good source of examples related to VBA. The following models are located in the Smarts folder within your main Arena™ folder (e.g., Program Files Rockwell Software Arena™ Smarts).

- Smarts 001: VBA-VariableArray Value
- Smarts 016: Displaying a Userform
- Smarts 017: Interacting with Variables
- Smarts 024: Placing Modules and Filling in Data
- Smarts 028: Userform Interaction
- Smarts 081: Using a Shape Object
- Smarts 083: Ending a Model Run
- Smarts 086: Creating and Editing Resource Pictures
- Smarts 090: Manipulating a Module's Repeat Groups
- Smarts 091: Creating Nested Submodels Via VBA
- Smarts 098: Manipulating Named Views
- Smarts 099: Populating a Module's Repeat Group
- Smarts 100: Reading in Data from Excel
- Smarts 109: Accessing Information
- Smarts 121: Deleting a Module
- Smarts 132: Executing Module Data Transfer
- Smarts 142: VBA Submodels
- Smarts 143: VBA-Animation Status Variables
- Smarts 155: Changing and Editing Global Pictures
- Smarts 156: Grouping Objects
- Smarts 159: Changing an Entity Attribute
- Smarts 161: User Function
- Smarts 166: Inserting Entities into a Queue
- Smarts 167: Changing an Entity Picture
- Smarts 174: Reading/Writing Excel Using VBA Module



**Figure 10.86** VBA example model.

- Smarts 175: VBA Builds and Runs a Model
- Smarts 176: Manipulating Arrays
- Smarts 179: Playing Multimedia Files Within a Model
- Smarts 182: Changing Model Data Interactively

The Smarts files 001, 024, 090, 099, and 109 are especially illuminating. In the next example, the UserFunction event and the VBA block are illustrated.

**10.7.3.2 The VBA Module and User-Defined Functions** This example presents how to (i) use a user form to set the value of a variable, (ii) call a user-defined function which uses the value of an attribute to compute a quantity, and (iii) use a VBA block to display information at a particular point in the model. Since the purpose of this example is to illustrate VBA, the model is a simple single-server queuing system as illustrated in Figure 10.86. The model consists of CREATE, ASSIGN, PROCESS, VBA, and DISPOSE modules.

The VBA block is found on the Blocks panel. To attach the Blocks panel, you can right-click in the Basic Process panel area and select Attach and then find the *Blocks.tpo* file. Now you are ready to lay down the modules as shown in Figure 10.86. The information for each module is given as follows:

- CREATE: Choose Random(expo) with mean time between arrivals of 1 hour and set the maximum number of entities to 5.
- ASSIGN: Use an attribute called *myPT* and assign a  $U(10, 20)$  random number to it via the UNIF(10,20) function.
- PROCESS: Use the SEIZE, DELAY, RELEASE option. Define a resource and seize 1 unit of the resource. In the Delay Type area, choose Expression and type in UF(1) for the expression.
- Using the VARIABLE data module to define a variable called *vPTFactor* and initialize it with the value 1.0.

No editing is necessary for the VBA block and the DISPOSE module. If you have any difficulty in completing these steps, you can look at the module dialogues in the Arena™ file called *Ch10VBAExample.doe*.

Now you are ready to enter the world of VBA. Using Alt-F11, open the VBA Editor. Double-click on the *ThisDocument* object and enter the code as shown in Listing 10.1. If you don't want to retype this code, then you can access it in the file *Ch10VBAExample.doe*.

### **Listing 10.1 Defining Variables in VBA**

```

Option Explicit

" Declare global object variables to refer
" to the Model and SIMAN objects
" Public allows them to be accessed anywhere in this module
" or any other vba module
Public gModelObj As Model
Public gSIMANObj As SIMAN

" Variables can be accessed via their uniquely assigned
" symbol number. An integer variable is needed to hold the index
" It is declared public here so it can be used throughout this
module
" and other vba modules
Public vPTFactorIndex As Integer

" Index for the myPT attribute
" It is declared private here so it can be used throughout this
module
Private myPTIndex As Integer

```

Let's walk through what this code means. The two public variables *gModelObj* and *gSIMANObj* are object reference variables of type Model and SIMAN, respectively. These variables allow access to the properties and methods of these two objects once the object references have been set. The Model object is part of Arena's VBA Object model and allows access to the features of the Model as a whole. The details of Arena's Object model can be found within the Arena™ help system by searching on *Automation Programmer's Reference*. The SIMAN object is created after the simulation has been complied and essentially gives access to the underlying simulation engine. The variables *vPTFactorIndex* and *myPTIndex* will be used to index into SIMAN to access the Arena™ variable *vPTFactor* and the attribute *myPT*.

This example will use VBA forms to get some input from the user and to also display some information. Thus, the forms to be used in the example need to be developed. Use Insert > UserForm to create two forms called Interact and UserForm1 as shown in Figure 10.87.

Use the show toolbox button to show the VBA controls toolbox. Then, you can select the desired control and place your cursor on the form at the desired location to complete the action. Build the forms as shown in the Figures 10.87 and 10.88. To place the labels used in the forms, use the label control (right next to the text box control on the Toolbox). The name of a form can be changed in the Misc > (Name) property as shown in Figure 10.89. Now that the forms have been built, the controls on the forms can be referenced within other VBA modules.

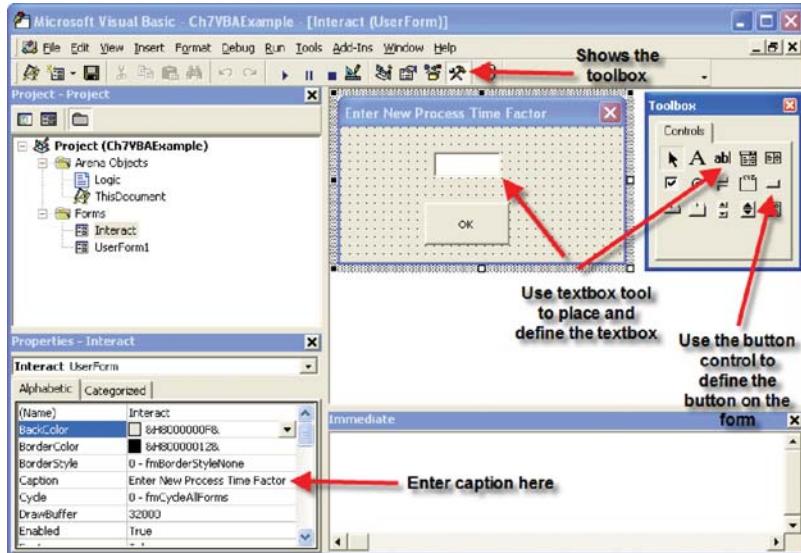


Figure 10.87 Building the interact form.

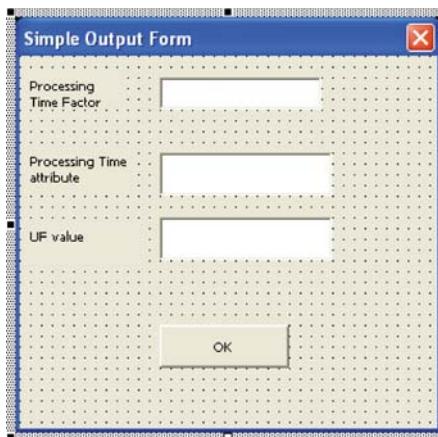
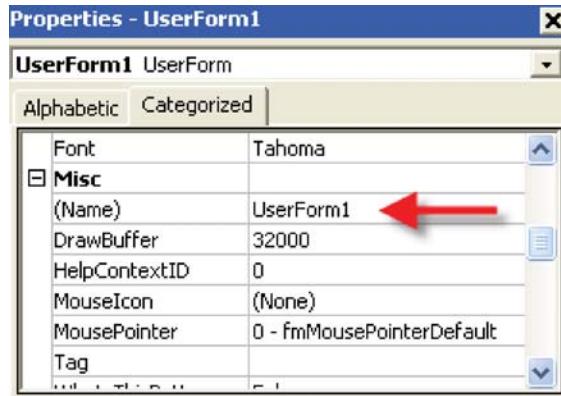


Figure 10.88 VBA UserForm1.

Before looking at the code for this situation, we need to understand how to interchange data between the Arena™ model and the VBA code. This will be accomplished using the SIMAN object within the VBA environment. When a model is complied, each element of the model is numbered. This is called the symbol number. The method, *SymbolNumber("element name")*, on the SIMAN object will return the index symbol number that can be used as an identifier for the named element. To find information on the SIMAN object, search the Arena™ help system on SIMAN Object (Automation). There are many properties and methods associated with the SIMAN Object. The Arena™ documentation states the following concerning the *SymbolNumber* method.



**Figure 10.89** Properties window.

#### *SymbolNumber Method*

**Syntax** `SymbolNumber (symbolString As String, index1 As Long, index2 As Long) As Long`

**Description** All defined simulation elements have a unique number. For those constructs that have names, this function may be used to return the number corresponding to the construct name. For example, many of the methods in the SIMAN Object require an argument like `resourceNumber` or `variableNumber` to identify the specific element. Since it is more common to know the name rather than the number of the item, `SymbolNumber("MyElementName")` is often used for the `elementNumber` type argument.

As indicated in the documentation, an important part of utilizing the SIMAN object's other methods is to have access to an Arena™ element's unique symbol number. The symbol number is an integer and is then used as an index into other method calls to identify the element of interest. According to the help system, the `VariableArrayValue` property can be used either to get or to set the value associated with the variable identified by the index number.

#### *VariableArrayValue Read/Write Property*

**Syntax** `VariableArrayValue (variableNumber As Long) As Double`

**Description** Gets/Sets the value of the variable, where `variableNumber` is the instance number of the variable as listed in the VARIABLES element. For more information on working with variables in automation, please see making variable assignments.

Now we are ready to look at the code. Listing 10.2 shows the VBA code for the `RunBeginSimulation` event. When the user executes the simulation, the `RunBeginSimulation` event is fired. The first two lines of the routine set the object references to the Model object and to the SIMAN object in order to store the values in the global variables that were previously defined for the VB module. The SIMAN object can then be accessed through the variable `gSIMANObj` to get the indexes of the attribute and variable `myPT` and `vPTFactor`. In the listing, after the indexes to the `myPT` and `vPTFactor` are found, the SIMAN object is used to access the `VariableArrayValue` property. In this code, the value of the variable is accessed

and then assigned to the value of the *TextBox1* control on the Interact form. Then, the Interact form is told to show itself and the focus is set to the text box, *TextBox1*, on the form for user entry.

### **Listing 10.2 RunBeginSimulation Event Routine**

```

Private Sub ModelLogic_RunBeginSimulation()
    " set object references
    Set gModelObj = ThisDocument.Model
    Set gSIMANObj = ThisDocument.Model.SIMAN

    " Get the index to the symbol number
    " if the symbol does not exist there will be an error,
    " There is no error handling in this example
    " The SymbolNumber method of the SIMAN object returns
    " the index associated with the named symbol

    " get the index for the myPT attribute
    myPTIndex = gSIMANObj.SymbolNumber("myPT")

    " get the index for the vPTFactor variable
    vPTFactorIndex = gSIMANObj.SymbolNumber("vPTFactor")

    " set the current value of the textbox to the
    " current value of the vPT variable in Arena
    " The VariableArrayValue method of the SIMAN object
    " returns the value of the variable associated with the index
    Interact.TextBox1.value =
        gSIMANObj.VariableArrayValue(vPTFactorIndex)

    " Display the user form
    Interact.Show
    " Set the focus to the textbox
    Interact.TextBox1.SetFocus

    " The code for setting the variable's value is
    " in the OK button user form code
End Sub

```

When the Interact form is displayed, the text box will show the current value of the variable. The user can then enter a new value in the text box and press the OK button. Since the user interacted with the OK button, the button press event within the VBA form code can be used to change the value of the variable within Arena™ to what was entered by the user.

To enter the code to react to the button press go to the Interact form and select the OK button on the form. Right-click on the button and from the context menu, select View Code. This will open up the form's module and create a VBA event to react to the button click.

The code provided in Listing 10.3 shows the use of the *VariableArrayValue* property to assign the value entered into the text box to the *vPTFactor* variable. To access a public variable within the *ThisDocument* module, you precede the variable with “*ThisDocument*”

(e.g., `ThisDocument.gSIMANObj`). Once the value of the text box has been assigned to the variable, the form is hidden and input focus is given back to the model via the `ThisDocument.gModelObj.Activate` method.

### **Listing 10.3 VBA Code for OK Button**

```

Private Sub CommandButton2_Click()
    " when the user clicks the ok button
    " Set the current value of vPT to the value
    " currently in the textbox
    " uses the global variable vPTFactorIndex
    " defined in module ThisDocument
    " uses the global variable for the SIMAN object
    " defined in ThisDocument

    ThisDocument.gSIMANObj.VariableArrayValue
    (ThisDocument.vPTFactorIndex) = TextBox1.value

    " hide the form
    Interact.hide
    " Tell the model to resume running
    ThisDocument.gModelObj.Activate
End Sub

```

Since the setting of the variable `vPTFactor` occurs as a result of the code initiated by the `RunBeginSimulation` event the value supplied by the user will be used for the entire run (unless changed again within the model or within VBA).

The model will now begin to create entities and execute as a normal Arena<sup>TM</sup> model. Each entity that is created will go through the ASSIGN module and have its `myPT` attribute set to a randomly drawn  $U(10, 20)$  number. Then the entity proceeds to the PROCESS module where it invokes the SEIZE, DELAY, and RELEASE logic. In this module, the processing time is specified by a user-defined function via the `UF(fid)` function. The `UF(fid)` function will call associated VBA code from directly within an Arena<sup>TM</sup> model.

You should think of the `UF(fid)` function as a mechanism by which you can easily call VBA functions. The `fid` argument is an integer that will be passed into VBA. This argument can be used to select from other user-written functions as necessary. Listing 10.3 shows the code for the UF function for this example. When you use the UF function, you must write your own code. To write the UF function, use the drop down box that lists the available VBA events on the `ThisDocument` module and select the `UserFunction` event. The `UserFunction` event routine that is created has two arguments. The first argument is an identifier that represents the current active entity, and the second argument is what was supplied by the call to `UF(fid)`.

When you create the function, it will not have any code. You should organize your code in a manner similar to that shown in Listing 10.4. As can be seen in the exhibit, the supplied function identifier is used within a VBA select-case statement to select from other available user-written functions.

By using this select-case construct, you can easily define a variety of your own functions and call whichever function you need from Arena<sup>TM</sup> by supplying the appropriate function identifier.

**Listing 10.4 VBA UserFunction() Event Routine**

```

" This Function allows you to pass a user
" defined value back to the
" module which called upon the UF(functionID) function in Arena.
" Use the functionID to select the function that you want via
" the case statement, add additional functions as necessary
" The functions can obviously be named something
" more useful than UserFunctionX()
" The functions must return a double
Private Function ModelLogic_UserFunction(ByVal entityID As Long,
 ByVal functionID As Long) As Double
    " entityID is the active entity
    " functionID is supplied when the user calls UF(functionID)

    Select Case functionID
        Case 1
            ModelLogic_UserFunction = UserFunction1()
        Case 2
            ModelLogic_UserFunction = UserFunction2()
    End Select

End Function

```

In order to implement the called functions, we need to understand how to access attributes associated with the active entity. This can be accomplished using two methods associated with the SIMAN object: *ActiveEntity* and *AttributeValue*. The help system describes the use of these methods as follows.

***ActiveEntity Method***

**Syntax** *ActiveEntity () As Long*

**Description** Returns the record location (the entity pointer) of the currently active entity, or 0 if there is not 1. This is particularly useful in a VBA block Fire event to access attributes of the entity that entered the VBA block in the model.

***AttributeValue Method***

**Syntax** *AttributeValue (entityLocation As Long, attributeNumber As Long, index1 As Long, index2 As Long) As Double*

**Description** Returns the value of general-purpose attribute *attributeNumber* with associated indices *index1* and *index2*. The number of indices specified must match the number defined for the attribute.

Let us see how to put these methods into action within the user-defined functions. Listing 10.5 shows how to access the value of an attribute associated with the active entity. First, the *ActiveEntity* property of the SIMAN object is used to get an identifier for the current active entity (i.e., the entity that entered the VBA block). Then, the *AttributeValue* method of the SIMAN object is used to get the value of the attribute.

**Listing 10.5 Implementing a User Defined Function**

```

Private Function UserFunction1() As Double
    " each entity has a unique id
    Dim activeEntityID As Long
    " get the number of the active entity
    " this could have been passed from ModelLogic_UserFunction
    activeEntityID = gSIMANObj.ActiveEntity

    Dim PTvalue As Double
    " get the value of the myPT attribute
    PTvalue = gSIMANObj.AttributeValue(activeEntityID,
    myPTIndex, 0, 0)

    Dim factor As Double
    factor = gSIMANObj.VariableArrayValue(vPTFactorIndex)

    " this could be complicated function of the attribute/
    variables
    " here it is very simple (and could have been done within
    Arena itself
    " rather than VBA
    UserFunction1 = PTvalue * factor
End Function

```

Within Arena<sup>TM</sup>, attributes can be defined as multidimensional via the ATTRIBUTES module. Multidimensional attributes are not discussed in this text, but the *AttributeValue* method allows for this possibility. The two indices that can be supplied indicate to the SIMAN object how to access the attribute. In the example, these two values are zero, which indicates that this is not a multidimensional attribute. Once the values of the *myPT* attribute and the *vPTFactor* variable are retrieved from the SIMAN object, they are used to compute the value that is returned by the UF function. The variables *factor* and *PTvalue* are used to calculate the product of their values. Admittedly, this calculation can be more readily accomplished directly in Arena<sup>TM</sup> without VBA, but the point is you can implement any complicated calculation using this architecture.

With the *UF* function, you can easily invoke VBA code from essentially any place within your Arena<sup>TM</sup> model. The *UF* function is especially useful for returning a value back to Arena<sup>TM</sup>. Using the VBA block, you can also invoke specific VBA code when the entity passes through the block. In this example, after the entity exits the PROCESS module, the entity enters a VBA block.

When you use a VBA block within your Arena<sup>TM</sup> model, each VBA block is given a unique number. Then, within the *ThisDocument* module, an individual event routine that is associated with each VBA block can be created. In the example, the VBA block is used to open up a form and display some information about the entity when it reaches the VBA block. This might be useful to do when running an Arena<sup>TM</sup> model in order to stop the model execution each time the entity goes through the VBA block to allow the user to interact with the form. However, most of the time the VBA block is used to execute complicated code that depends on the state of the system when the entity passes through the VBA block. Listing 10.6 shows the code for the VBA block event.

**Listing 10.6 Example VBA Block Event**

```
Private Sub VBA_Block_1_Fire()
    " set the values of the textboxes
    UserForm1.TextBox1.value = gSIMANObj.VariableArrayValue
    (vPTFactorIndex)
    UserForm1.TextBox2.value = gSIMANObj.AttributeValue(gSIMANObj.
        ActiveEntity, myPTIndex, 0, 0)
    UserForm1.TextBox3.value = UserFunction1()
    " Display the user form
    UserForm1.Show
End Sub
```

By now, this code should start to look familiar. The SIMAN object is used to access the values of the attribute and the variable that are of interest and then set them equal to the values for the *text boxes* that are being used on the form. Then, the form is shown. This brings up the form so that the user can see the values. In *UserForm1*, a command button was defined to close the form. The logic for hiding the form is shown in Listing 10.7.

**Listing 10.7 Hiding the User Form**

```
Private Sub CommandButton1_Click()
    " hide the form
    UserForm1.hide
    " Tell the model to resume running
    ThisDocument.gModelObj.Activate
End Sub
```

The show and hide functionality of VBA forms are used within this example so that new instances of the forms did not have to be created each time they are used. This keeps the form in memory so that the controls on the form can be readily accessed. This is not necessarily the best practice for managing the forms, but allows the discussion to be simplified. As long as you do not have a large number of forms, this approach is reasonable. In addition, within the examples, there is no error catching logic. VBA has a very useful error catching mechanism and professional code should check for and catch errors.

**10.7.4 Generating Correlated Random Variates**

The final example involves how to explicitly model dependence (correlation) within input distributions. In the fitting of input models, it was assumed and tested that the sample observations did not have correlation. But, what do you do if the data does have correlation? For example, let  $X_i$  be the service time of the  $i$ th customer. What if the  $X_i$  have significant correlation? That is, the service times are correlated. Arrival processes might also be correlated. That is, the time between arrivals might be correlated. Research has shown, see

Patuwo et al. [1993] and Livny et al. [1993], that ignoring the correlation when it is in fact present can lead to gross underestimation of the actual performance estimates for the system.

The discussion here is based on the normal-to-anything (NORTA) Transformation as discussed in Banks et al. [2005], Cario and Nelson [1998], Cario and Nelson [1996], and Biller and Nelson [2005]. Suppose you have a  $N(0, 1)$  random variable,  $Z_i$ , and a way to compute the cumulative distribution function (CDF),  $\Phi(z)$ , of the normal distribution. Then, based on the inverse transform technique, the random variable,  $\Phi(Z_i)$ , will have a  $U(0, 1)$  distribution. Suppose that you wanted to generate a random variable  $X_i$  with CDF  $F(x)$ , then you can use  $\Phi(Z_i)$  as the source of uniform random numbers in an inverse transform technique.

$$X_i = F^{-1}(U_i) = F^{-1}((\Phi(Z_i))) \quad (10.9)$$

This transform is called the NORTA transformation. It can be shown that even if the  $Z_i$  are correlated (and thus so are the  $\Phi(Z_i)$ ), then the  $X_i$  will have the correct CDF and will also be correlated. Unfortunately, the correlation is not directly preserved in the transformation so that if the  $Z_i$  have correlation  $\rho_z$ , then the  $X_i$  will have  $\rho_x$  (not necessarily the same). Thus, in order to induce correlation in the  $X_i$ , you must have a method to induce correlation in the  $Z_i$ .

One method to induce correlation in the  $Z_i$  is to generate the  $Z_i$  from an autoregressive time-series model of order 1, that is, AR(1). An AR(1) model with  $N(0, 1)$  marginal distributions has the following form:

$$Z_i = \phi Z_{i-1} + \varepsilon_i \quad (10.10)$$

where  $Z_1 \sim N(0, 1)$ , with independent and identically normally distributed errors,  $\varepsilon_i \sim N(0, 1 - \phi^2)$  with  $-1 < \phi < 1$  for  $i = 2, 3, \dots$

It is relatively straightforward to generate this process by first generating  $Z_1$ , then generating  $\varepsilon_i \sim N(0, 1 - \phi^2)$  and using  $Z_i = \phi Z_{i-1} + \varepsilon_i$  to compute the next  $Z_i$ . It can be shown that this AR(1) process will have lag-1 correlation:

$$\phi = \rho^1 = \text{corr}(Z_i, Z_{i+1}) \quad (10.11)$$

Therefore, you can generate a random variable  $Z_i$  that has a desired correlation, and through the NORTA transformation, produce  $X_i$  that are correlated with the correlation being *functionally* related to  $\rho_z$ . By changing  $\phi$ , one can get the correlation for the  $X_i$  that is desired. Procedures for accomplishing this are given in the previously mentioned references. The spreadsheet *NORTAExample.xls* can also be used to perform this search process.

The implementation of this technique cannot be readily achieved in a general way within Arena™ through the use of standard modules. In order to implement this method, you can make use of VBA. The model *NORTA-VBA.doe* shows how to implement the NORTA algorithm with a user-defined function. Just as illustrated in the last section, a user-defined function can be written as shown in Listing 10.8.

**Listing 10.8 VBA User Function Event**

```

Private Function ModelLogic_UserFunction(ByVal entityID As Long,
ByVal functionID As Long) As Double

    Select Case functionID
        Case 1
            ModelLogic_UserFunction = CorrelatedUniform()
        Case 2
            Dim u As Double
            u = CorrelatedUniform()
            Dim x As Double
            x = expoInvCDF(u, 2)
            ModelLogic_UserFunction = x
    End Select

End Function

```

When you uses UF(1), a correlated  $U(0, 1)$  random variable will be returned. If the user uses UF(2), then a correlated exponential distribution with a mean of 2 will be returned. Notice that in generating the correlated exponential random variable, a correlated uniform is first generated and then used in the inverse transform method to transform the uniform to the proper distribution. Thus, provided that you have functions that implement the inverse transform for the desired distributions, you can generate correlated random variables.

Listing 10.9 illustrates how the NORTA technique is used to generate the correlated uniform random variables. The variable *phi* is used to control the resulting correlation in the AR(1) process. The function *SampleNormal* associated with the SIMAN object is used to generate a  $N(0, 1)$  random variable that represents the error in the AR(1) process.

**Listing 10.9 VBA Function to Generate Correlated Uniforms**

```

Private Function CorrelatedUniform() As Double
    Dim phi As Double
    Dim e As Double
    Dim u As Double
    " change phi inside this function to get a different
correlation
    phi = 0.8
    " generate error
    e = gSimanObj.SampleNormal(0, 1 - (phi * phi), 1)
    " compute next AR(1) Z
    mZ = phi * mZ + e
    " use normal cdf to get uniform
    u = NORMDIST(mZ)
    CorrelatedUniform = u
End Function

Private Function expoInvCDF(u As Double, mean As Double) As
Double
    expoInvCDF = -mean * Log(1 - u)
End Function

```

The variable  $mZ$  has been defined at the VBA module level, and thus it retains its value between invocations of the *CorrelatedUniform* function. The variable  $mZ$  represents the AR(1) process. The current value of  $mZ$  is multiplied with  $\phi$  and the error is added to obtain the next value of  $mZ$ . The initial value of  $mZ$  was determined by implementing the *RunBeginReplication* event within the *ThisDocument* module. Because of the NORTA transformation,  $mZ$  is  $N(0, 1)$ , and the function *NORMDIST* is used to compute the probability associated with the supplied  $z$ -value. The *NORMDIST* function (not shown here) is an implementation of the CDF for the standard normal distribution.

With minor changes, the code supplied in *NORTA-VBA.doe* can be easily adapted to generate other correlated random variables. In addition, the use of the UF function can be expanded to generate other distributions that Arena<sup>TM</sup> does not have built in (e.g., binomial).

Arena<sup>TM</sup> is a very capable language, but occasionally, you will need to access the full capabilities of a standard programming language. This section illustrated how to utilize VBA within the Arena<sup>TM</sup> environment by either using the predefined automation events or by defining your own functions or events via the UF function and the VBA block. With these constructs you can make Arena<sup>TM</sup> into a powerful tool for end users. There is one caveat with respect to VBA integration that must be mentioned. Since VBA is an interpreted language, its execution speed can be slower than compiled code. The use of VBA within Arena<sup>TM</sup> as indicated within this section can increase the execution time of your models. If execution time is a key factor for your application, then you should consider using C/C++ rather than VBA for your advanced integration needs. Arena<sup>TM</sup> allows access to the SIMAN runtime engine via C language functions. Essentially, you write your C functions and bundle them into a dynamic-linked library which can be linked to Arena<sup>TM</sup>. For more information on this topic, you should search the Arena<sup>TM</sup> help system under *Introduction to C Support*.

## 10.8 SUMMARY

This chapter provided a discussion of miscellaneous modeling constructs that can improve your Arena<sup>TM</sup> modeling. The modeling of non-stationary arrivals was discussed, and it motivated the exploration of advanced resource modeling constructs. The advanced resource modeling constructs allow for resources to be subjected to either scheduled or random capacity changes. Then, how Arena<sup>TM</sup> tabulates entity and resource costs were presented. The PICKSTATION allows for an entity to use criteria to select from a set of listed stations. In addition, the PICKUP and DROPOFF modules provide another mechanism by which entities can be added to or removed from the active entity's group. Finally, some programming aspects of Arena<sup>TM</sup> (including generic stations and VBA) that can make your modeling more productive were presented. In the next chapter, a comprehensive example is presented so that you can experience the full range of applying Arena<sup>TM</sup> to a simulation problem.

## EXERCISES

- 10.1 Customers arrive at a one-window drive in bank according to a Poisson distribution with a mean of 10 per hour. The service time for each customer is exponentially distributed with a mean of 5 minutes. There are three spaces in front of the window including that for the car being served. Other arriving cars can wait outside these

three spaces. Use the frequency option of the STATISTIC module to the probability that an arriving customer can enter one of the three spaces in front of the window. Compare your results with your answers to Exercise 8.5.

- 10.2 This problem analyzes the cost associated with the LOTR Ring Maker, Inc. configurations described in Example 7.5. In particular, the master ring maker cost \$30 per hour whether he/she is busy or idle. His/her time is considered value added. The rework craftsman earns \$22 per hour; however, his/her time is considered nonvalue added since it could be avoided if there were no rework. Both the master ring maker and the rework craftsman earn \$0.50 for each ring that they work on. The inspector earns \$15 per hour without any piece work considerations, and the packer earns \$10 per hour (again with no piece work). The inspection time is considered nonvalue added; however, the packing time is considered value added. The work rules for the workers include that they should get a 10-minute break after 2 hours into their shift, a 30-minute lunch after 4 hours into their shift, a 10-minute break 2 hours after the completion of lunch, and a 15-minute clean up allowance before the end of the shift.

While many would consider a magic ring priceless, LOTR Ring Maker, Inc. has valued a pair of rings at approximately \$10,000. The holding charge for a ring is about 15% per year. Assume that there are 365 days in a year and that each day has 24 hours when determining the holding cost per hour for the pair of rings. Simulate configuration 1 and configuration 2 as given in Chapter 7 and report the entity and resource costs for the configurations. Which configuration would you recommend?

- 10.3 Reconsider the machine interference problem from Chapter 8. Use the costing functionality discussed in this chapter and find the best assignment of operators in terms of cost while maintaining an average machine utilization of over 70%
- 10.4 The Super Ready Auto Club has been serving customers for many years. The Super Ready Auto Club provides club, travel, and financial services to its members. One of its most popular services includes auto touring and emergency road service. Travel provides cruise packages, airline travel, and other vacation packages with special discounts for members. Finally, the financial services issue credit cards for members at special interest rates.

Super Ready Auto Club has regional call centers that handle incoming calls from members within a given region of the country. Table 10.8 indicates the mean of the number of calls, recorded on an hourly basis over a period of 7 days, aggregated over a three-state region.

The three types of service calls occur with 60% for auto service, 35% for credit card services, and 15% for travel services during the 8 a.m. to 8 p.m. time frame. For the other hours of the day, the proportion changes to 90% for auto service, 10% for credit card services, and 0% for travel services. A sample of call service times were recorded for each of the types as shown in Tables 10.9–10.11.

Call center operators cost \$18 per hour including fringe benefits. It is important that the probability of a call waiting more than 3 minutes for an operator could be less than 10%. Find a minimum cost staffing plan for each hour of the day for each day of the week that on-average meets the call probability waiting criteria. Measure the waiting time for a call, the average number of calls waiting, and the utilization of the operators. In addition, measure the waiting time of the calls by type.

- (a) What happens to the waiting times if road-side assistance calls are given priority over the credit card and travel service calls?

**TABLE 10.8 Mean Number of Calls Received for each Hour**

Hour	Mon	Tue	Wed	Thur	Fri	Sat	Sun
1	6.7	6.1	8.7	8.9	5.6	6.2	9.4
2	9.7	9.4	9.3	7.1	1.4	7.0	8.8
3	8.5	6.0	70.4	8.6	7.9	7.8	13.4
4	9.0	10.0	6.8	8.3	6.4	6.3	6.7
5	6.8	9.2	10.9	44.9	8.2	6.5	7.5
6	8.5	4.2	1.4	6.3	7.5	7.8	8.0
7	6.7	6.9	9.0	8.9	6.7	46.2	9.8
8	38.2	35.0	75.8	57.8	37.2	82.5	78.8
9	20.0	77.6	48.7	25.9	67.1	28.1	99.2
10	76.2	75.4	71.1	51.3	86.2	106.9	42.0
11	92.5	105.4	28.2	100.2	90.9	92.4	43.6
12	76.4	37.3	37.3	77.8	60.3	37.9	68.9
13	79.1	64.2	25.4	76.8	65.1	77.9	116.6
14	75.2	102.2	64.1	48.7	93.3	64.5	39.5
15	117.9	26.8	43.8	80.1	86.4	96.8	52.1
16	100.2	85.9	54.3	144.7	70.9	167.9	153.5
17	52.7	64.7	94.7	94.6	152.2	98.0	63.7
18	65.3	114.9	124.9	213.9	91.5	135.7	79.8
19	126.1	69.6	89.4	43.7	62.7	111.9	180.5
20	116.6	70.2	116.8	99.6	113.0	84.3	64.1
21	38.6	20.5	7.6	3.8	58.6	50.8	68.5
22	8.5	17.1	8.8	13.5	8.2	68.1	7.8
23	9.8	48.9	8.9	95.4	40.8	58.5	44.1
24	9.6	1.3	28.5	9.6	32.2	74.5	46.7

**TABLE 10.9 Road-Side Service Call Times in Minutes**

33.6	34.0	33.2	31.3	32.3	32.1	39.0	35.2	37.6	37.4
32.3	37.2	39.3	32.1	34.7	35.1	33.6	32.8	32.8	40.0
37.8	38.0	42.4	37.4	36.6	36.6	33.3	34.7	30.2	33.1
36.8	34.4	36.4	35.9	32.6	37.6	36.7	32.3	40.0	37.0
36.0	34.6	33.9	31.7	33.8	39.3	37.8	33.7	35.2	38.2
34.6	33.5	36.3	38.9	35.5	35.2	35.0	33.8	35.8	35.8
38.2	38.8	35.7	38.7	30.0	33.6	33.4	34.7	35.1	35.5
34.0	33.2	33.7	32.5	28.9	34.4	34.2	31.4	38.7	35.3
35.5	39.4	32.6	36.2	33.2	39.3	41.1	34.3	38.6	33.0
35.3	38.1	33.5	34.0	36.4	33.6	43.1	37.2	35.6	36.0

- (b) Consider how you would handle the following work rules. The operators should get a 10-minute break every 2 hours and a 30-minute food/beverage break every 4 hours. Discuss how you would staff the center under these conditions ensuring that there is always someone present (i.e., they are not all on break at the same time).

- 10.5 The test and repair system described in Chapter 8 has three testing stations, a diagnostic station, and a repair station. Suppose that the machines at the test station are subject to random failures. The time between failures is distributed according to

**TABLE 10.10** Vacation Package Service Call Times in Minutes

52.2	23.7	33.9	30.5	18.2	45.2	38.4	49.8	53.9	38.8
33.4	44.2	28.4	49.0	24.9	46.4	35.8	40.3	16.3	41.4
63.1	37.5	33.5	48.0	27.6	38.2	28.6	35.2	24.5	42.9
38.9	19.8	32.7	41.4	42.7	27.4	38.7	30.1	39.6	53.5
28.8	42.2	42.4	29.2	22.7	50.9	34.2	53.1	18.5	26.5
40.9	20.7	36.6	33.7	26.4	32.8	25.1	39.7	30.9	34.2
35.5	31.8	27.2	28.6	26.9	30.6	26.1	23.2	33.3	35.0
29.6	31.6	29.9	28.5	29.3	30.6	31.2	26.7	25.7	41.2
27.5	52.0	27.3	69.0	34.2	31.4	21.9	29.8	31.4	46.1
23.8	35.9	41.5	51.0	17.9	33.9	32.2	26.5	25.0	54.6

**TABLE 10.11** Credit Card Service Call Times in Minutes

13.5	10.1	14.4	13.9	14.2	11.4	11.0	14.4	12.0	14.4
14.4	13.5	13.4	14.5	14.8	14.9	13.2	11.0	14.9	15.0
14.7	14.3	12.3	14.8	12.4	14.9	15.0	14.0	14.6	14.6
15.0	10.3	11.3	15.0	15.0	15.0	14.9	14.8	11.5	11.9
13.5	14.9	11.0	10.8	15.0	13.8	13.8	14.5	14.3	14.0
14.9	10.9	13.9	13.5	15.0	14.4	13.2	15.0	14.5	14.4
15.0	14.5	14.5	14.9	11.6	13.6	12.4	11.9	11.6	15.0
11.1	13.7	13.9	14.6	11.5	14.4	15.0	10.7	10.4	14.5
12.3	13.8	14.7	13.8	14.7	14.8	14.9	14.2	12.9	14.3
14.5	11.5	12.1	14.3	13.6	14.1	12.6	11.7	14.7	13.4

an exponential distribution with a mean of 240 minutes and is based on busy time. Whenever a test station fails, the testing software must be reloaded, which takes between 10 and 15 minutes uniformly distributed. The diagnostic machine is also subject to usage failures. The number of uses to failure is distributed according to a geometric distribution with a mean of 100. The time to repair the diagnostic machine is uniformly distributed in the range of 15 to 25 minutes. Examine the effect of explicitly modeling the machine failures for this system in terms of risks associated with meeting the terms of the contract.

- 10.6 A proposal for a remodeled Sly's Convenience Store has six gas pumps, each having their own waiting line. The interarrival time distribution of the arriving cars is Poisson with a mean of 60 per hour. The time to fill up and pay for the purchase is exponentially distributed with a mean of 6 minutes. Arriving customers choose the pump that has the least number of cars (waiting or using the pump). Each pump has room to handle three cars (one in service and two waiting). Cars that cannot get a pump wait in an overall line to enter a line for a pump. The long-run performance of the system is of interest.
- (a) Estimate the average time in the system for a customer and the average number of customers waiting in each line, the overall line, and the system, and estimate the percentage of time that there are 1, 2, 3 customers waiting in each line. In addition, the percentage of time that there are 1, 2, 3, 4 or more customers waiting to enter a pump line should be estimated. Assume that each car is between

- 16 and 18 feet in length. About how much space in the overall line would you recommend for the convenience store.
- (b) Occasionally, a pump will fail. The time between pump failures is distributed according to an exponential distribution with a mean of 40 hours and is based on clock time (not busy time). The time to service a pump is exponentially distributed with a mean of 4 hours. For simplicity, assume that when a pump fails, the customer using the pump is able to complete service before the pump is unusable and ignore the fact that some customers may be in line when the pump fails. Simulate the system under these new conditions and estimate the same quantities as requested in part 10.6.a. In addition, estimate the percentage of time that each pump spends idle, busy, or failed. Make sure that newly arriving customers do not decide to enter the line of a pump that has failed.
- (c) The assumption that a person waiting in line when a pump fails stays in line is unrealistic. Still assume that a customer receiving service when the pump fails can complete service, but now allow waiting customers to exit their line and rejoin the overall line if a pump fails while they are in line. Compare the results of this new model with those of part 10.6.b.
- 10.7 Apply the concepts of generic station modeling to the system described in Exercise 9.14. Be sure to show that your results are essentially the same with and without generic modeling.
- 10.8 A single automatic guided vehicle (AGV) is used to pick up finished parts from three machines and drop them off at a store room. The AGV is designed to carry 5 totes. Each tote can hold up to 10 parts. The machines produce individual parts according to a Poisson process with the rates indicated in the table below. The machines are designed to directly drop the parts into a tote. When a tote is full, it is released down a gravity conveyor to the AGV loading area. It takes 2 seconds for the tote to move along the conveyor to the machine's loading area.

Station	Production Rate	Tote Loading Time (Seconds)	Travel Distance to Next Station (Meters)
Machine 1	2 per minute	uniform(3,5)	40
Machine 2	1 per minute	uniform(2,5)	28
Machine 3	3 per minute	uniform(4,6)	55

The AGV moves from station to station in the sequence (Machine 1, Machine 2, Machine 3, store room). If there are totes waiting at the machine's loading area, the AGV loads the totes up to its capacity. After the loading is completed, the AGV moves to the next station. If there are no totes waiting at the station, the AGV delays for 5 seconds, if any totes arrive during the delay, they are loaded; otherwise, the AGV proceeds to the next station. After visiting the third machine's station, the AGV proceeds to the store room. At the storeroom, any totes that the AGV is carrying are dropped off. The AGV is designed to release all totes at once, which takes between 8 and 12 seconds uniformly distributed. After completing its visit to the store room, it proceeds to the first machine. The distance from the storeroom to the first station is 30 meters. The AGV travels at a velocity of 30 meters per minute. Simulate the performance of this system over a period of 10,000 minutes.

- (a) Estimate the average system time for a part. What is the average number of totes carried by the AGV? What is the average route time? That is, what is the average time that it takes for the AGV to complete one circuit of its route?
- (b) Suppose the sizing of the loading area for the totes is of interest. What is the required size of the loading area (in terms of the number of totes) such that the probability that all totes can wait is virtually 100%.
- 10.9 Suppose the shuttle bus system described in Section 10.6.3 now has three bus stops. Passengers arrive to bus stop 1 according to a Poisson arrival process with a mean rate of 6 per hour. Passengers arrive to bus stop 2 according to a Poisson arrival process with a mean rate of 10 per hour. Passengers arrive to bus stop 3 according to a Poisson arrival process with a mean rate of 12 per hour. Thirty percent of customers arrive to stop 1 desire to go to stop 2 with the remaining going to stop 3. For those passengers arriving to stop 2, 75% want to go to stop 1 and 25% want to go to stop 3. Finally, for those passengers who originate at stop 3, 40% want to go to stop 1 and 60% want to go to stop 2. The shuttle bus now has 20 seats. The loading and unloading times per passenger are still the same as in the example and the travel time between stops is still the same. Simulate this system for 8 hours of operation and estimate the average waiting time of the passengers, the average system time of the passengers, and the average number of passengers left at the stop because the bus was full.
- 10.10 SQL queries arrive to a database server according to a Poisson process with a rate of 1 query every minute. The time that it takes to execute the query on the server is typically between 0.6 and 0.8 minutes uniformly distributed. The server can only execute 1 query at a time.
- (a) Develop a simulation model to estimate the average delay time for a query and the probability that there are 1, 2, 3 or more queries waiting.
- (b) Suppose that data on the interarrival times for the queries shows that the time between arrivals are still exponentially distributed with a mean of 1 minute but the arrival process is correlated with a lag-1 correlation of 0.85. Simulate this situation and compare your results to the results of part 10.10.a.
- 10.11 Section 10.7.2.4 discusses how to store the parameter values of replications in a Microsoft Access database and to read in those parameter values when running replications within Arena<sup>TM</sup>. In addition, it indicates how to write out the results to a Microsoft Access database. The discussion suggests that if each experimental parameter setting requires multiple replications to simply repeat those parameter settings within the Access data input table. This can be tedious for large experimental designs. This problem considers how to modify the basic logic to allow multiple replications for each design point without duplicating them in the input table. Consider the following pseudo-code:

```

CREATE file reading/writing entity
DECIDE
  IF (MOD(NREP-1,vMaxReps) == 0 THEN
    READ in and ASSIGN parameter values
  END IF
END DECIDE
DELAY until end of replication

```

WRITE out replication results  
 DISPOSE file reading/writing entity

The variable *vMaxReps* is the number of replications that you want each design point repeated. Implement and test this pseudo-code on the *PharmacyModelRWv1.doe* file with each design point being repeated three times.

- 10.12 Develop a spreadsheet that uses the following set of uniform random number to generate 18 correlated random variables using the NORTA technique with an AR(1) process with  $\phi = 0.7$  and a gamma distribution with alpha (shape) parameter equal to 2 and a beta (scale) parameter equal to 5. What is the resulting lag-1 correlation of the gamma random variables.

0.943	0.398	0.372	0.943	0.204	0.794
0.498	0.528	0.272	0.899	0.294	0.156
0.102	0.057	0.409	0.398	0.400	0.997

- 10.13 Consider the non-stationary arrival process example provided in Section 10.2 involving the pharmacy model. See Table 10.2 and Figure 10.1. Use the information provided to create an Arrivals SCHEDULE for the Pharmacy model. Run the model under the same conditions as those given in Chapter 4 and compare the results to those from Chapter 4. Chapter 4 presents steady-state results for the pharmacy model. Do these results apply to this situation? Why or why not?



---

# 11

---

## APPLICATION OF SIMULATION MODELING

### LEARNING OBJECTIVES

- To be able to understand the issues in developing and applying simulation to real systems.
- To be able to perform experiments and analysis on practical simulation models.

#### 11.1 INTRODUCTION

Chapter 1 presented a set of general steps for problem solving called DEGREE. Those steps were expanded into a methodology for problem solving within the context of simulation. As a reminder, the primary steps for a simulation study can be summarized as follows:

1. Problem formulation
  - (a) Define the system and the problem
  - (b) Establish performance metrics
  - (c) Build conceptual model
  - (d) Document modeling assumptions
2. Simulation model building
  - (a) Model translation

- (b) Input data modeling
- (c) Verification and validation
- 3. Experimental design and analysis
  - (a) Preliminary runs
  - (b) Final experiments
  - (c) Analysis of results
- 4. Evaluate and iterate
- 5. Documentation
- 6. Implementation of results

To some extent, our study of simulation has followed these general steps. Chapter 4 introduced a basic set of modeling questions and approaches that are designed to assist with step 1:

- *What is the system? What information is known by the system?*
- *What are the required performance measures?*
- *What are the entities? What information must be recorded or remembered for each entity? How should the entities be introduced into the system?*
- *What are the resources that are used by the entities? Which entities use which resources and how?*
- *What are the process flows? Sketch the process or make activity flow diagrams.*
- *Develop pseudo-code for the model.*

Answering these questions can be extremely helpful in developing a conceptual understanding of a problem in preparation for the simulation model building steps of the methodology.

Chapter 5 provided the primary modeling constructs to enable the translation of a conceptual model to a simulation model within Arena<sup>TM</sup>. Since conceptual models often involve randomness, Chapter 6 showed how input distributions can be formed and presented the major methods for obtaining random values to be used within a simulation. Since random inputs imply that the outputs from the simulation will also be random, Chapter 7 showed how to properly analyze the statistical aspects of simulation. This allows appropriate experimental analysis techniques to be successfully employed during a simulation project. Finally, Chapters 8-10 delved deeper into a variety of modeling situations to round out the tool set of modeling concepts and Arena<sup>TM</sup> constructs that you can bring to bear on a problem.

At this point, you have learned a great deal concerning the fundamentals of simulation modeling and analysis. The purpose of this chapter is to help you to put your new knowledge into practice by demonstrating the modeling and analysis of a system in its entirety. Ideally, experience in simulating a real system would maximize your understanding of what you have learned; however, a realistic case study should provide this experience within the limitations of the textbook. During the past decade, Rockwell Software (the makers of Arena<sup>TM</sup>) and the Institute of Industrial Engineers (IIE) have sponsored a student contest involving the use of Arena<sup>TM</sup> to model a realistic situation. The contest problems have been released to the public and can be found at [www.arenasimulation.com](http://www.arenasimulation.com). This chapter solves one of the previous contest problems. This process will illustrate the simulation modeling steps from

conceptualization to final recommendations. After solving the case study, you should have a better appreciation for and a better understanding of the entire simulation process.

## **11.2 SM TESTING CONTEST PROBLEM DESCRIPTION**

This section reproduces in its entirety the 7th Annual Contest Problem entitled “SM Testing.” Then, the following sections present a detailed solution to the problem. As you read through the following section, you should act as if you are going to be required to solve the problem. That is, try to apply the simulation modeling steps by jotting down your own initial solution to the problem. Then, as you study the detailed solution, you can compare your approach to the solution presented here.

### **Rockwell Software/IIE 7th Annual Contest Problem: SM Testing**

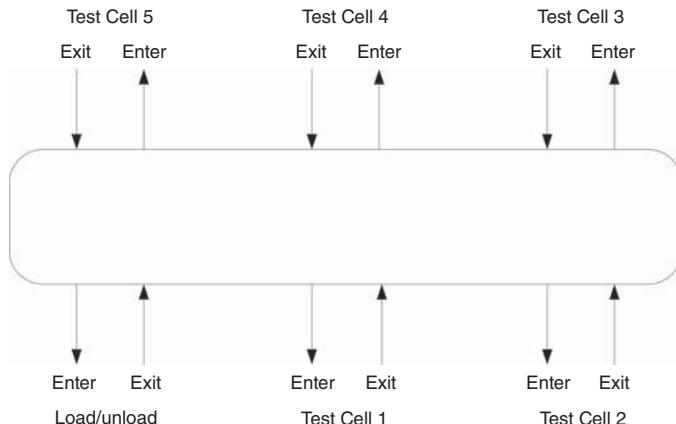
SM Testing is the parent company for a series of small medical laboratory testing facilities. These facilities are often located in or near hospitals or clinics. Many of them are new and came into being as a direct result of cost-cutting measures undertaken by the medical community. In many cases, the hospital or clinic bids their testing out to an external contractor but provides space for the required laboratory within their own facility.

SM Testing provides a wide variety of testing services and has long-term plans to increase our presence in this emerging market. Recently, we have concentrated on a specific type of testing laboratory and have undertaken a major project to automate most of these facilities. Several pilot facilities have been constructed and have proven to be effective not only in providing the desired service but also in their profitability. The current roadblock to a mass offering of these types of services is our inability to size the automated system properly to the specific site requirements. Although a method was developed for the pilot projects, it dramatically underestimated the size of the required system. Thus, additional equipment was required when capacity problems were uncovered. Although this trial-and-error approach eventually provided systems that were able to meet the customer requirements, it is not an acceptable approach for mass introduction of these automated systems. The elapsed time from when the systems were initially installed to when they were finally able to meet the customer demands ranged from 8 to 14 months. During that time, manual testing supplemented the capacity of the automated system. This proved to be extremely costly.

It is obvious that we could intentionally oversize the systems as a way of always meeting the projected customer demand, but it is understood that a design that always meets demand may result in a very expensive system with reduced profitability. We would like to be able to size these systems easily so that we meet or exceed customer requirements while keeping our investment at a minimum. We have explored several options to resolve this problem and have come to the conclusion that computer simulation may well provide the technology necessary to size these systems properly.

Prior to releasing this request for recommendations, our engineering staff developed a standard physical configuration that will be used for all future systems. A schematic of this standard configuration is shown in Figure 11.1.

The standard configuration consists of a transportation loop or racetrack joining six different primary locations: one load/unload area and five different test cells. Each testing cell will contain one or more automated testing devices that perform a test specific to that cell.



**Figure 11.1** Schematic of standard configuration.

The load/unload area provides the means for entering new samples into the system and removing completed samples from the system.

The transportation loop or racetrack can be visualized as a bucket conveyor or a simple power-and-free conveyor. Samples are transported through the system on special sample holders that can be thought of as small pallets or carts that hold the samples. The number of sample holders depends on the individual system. The transportation loop is 48 feet long, and it can accommodate a maximum of 48 sample holders, spaced at 1-foot increments. Note that because sample holders can also be within a test cell or the load/unload area, the total number of sample holders can exceed 48. The distance between the *Enter* and *Exit* points at the load/unload area and at each of the test cells is 3 feet. The distance between the *Exit* point of one cell and the *Enter* point of the next cell is 5 feet.

Let us walk through the movement of a typical sample through the system. Samples arriving at the laboratory initially undergo a manual preparation for the automated system. The sample is then placed in the input queue to the load/unload area. This manual preparation typically requires about 5 minutes. When the sample reaches the front of the queue, it waits until an empty sample holder is available. At that point, the sample is automatically loaded onto the sample holder, and the unit (sample on a sample holder) enters the transportation loop. The process of a unit entering the transportation loop is much like a car entering a freeway from a ramp. As soon as a vacant space is available, the unit (or car) merges into the flow of traffic. The transportation loop moves the units in a counterclockwise direction at a constant speed of 1 foot per second. There is no passing allowed.

Each sample is bar coded with a reference to the patient file as well as the sequence of tests that need to be performed. A sample will follow a specific sequence. For example, one sequence requires that the sample visit Test Cells 5-3-1 (in that order). Let us follow one of these samples and sample holders (units) through the entire sequence. It leaves Load/Unload at the position marked Exit and moves in a counterclockwise direction past Test Cells 1 through 4 until it arrives at the Enter point for Test Cell 5. As the unit moves through the system, the bar code is read at a series of points in order for the system to direct the units to the correct area automatically. When it reaches Test Cell 5, the system checks to see how many units are currently waiting for testing at Test Cell 5. There is only capacity for 3 units in front of the testers, regardless of the number of testers in the cell. This capacity is the same for all of the five test cells. The capacity (3) does not include any units currently being tested

or units that have completed testing and are waiting to merge back onto the transportation loop. If room is not available, the unit moves on and will make a complete loop until it returns to the desired cell. If capacity or room is available, the unit will automatically divert into the cell (much like exiting from a freeway). The time to merge onto or exit from the loop is negligible. A schematic of a typical test cell is shown in Figure 11.2.

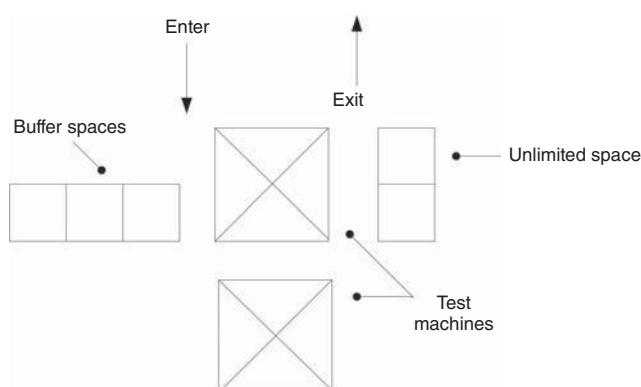
As soon as a tester becomes available, the unit is tested, the results are recorded, and the unit attempts to merge back onto the loop. Next it would travel to the Enter point for Test Cell 3, where the same logic is applied that was used for Test Cell 5. Once that test is complete, it is directed to Test Cell 1 for the last test. When all of the steps in the test sequence are complete, the unit is directed to the Enter point for the Unload area.

The data-collection system has been programmed to check the statistical validity of each test. This check is not performed until the sample leaves a tester. If the test results do not fall into accepted statistical norms, the sample is immediately sent back for the test to be performed a second time.

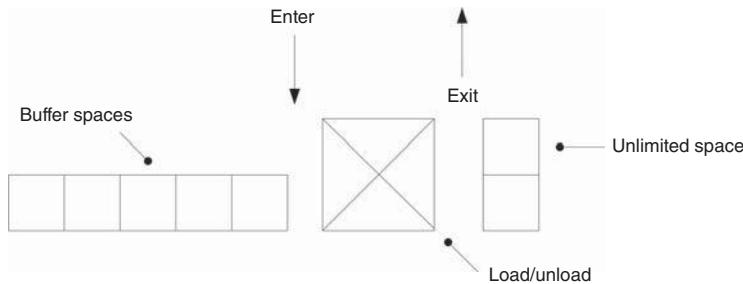
Although there can be a variable number of test machines at each of the test cells, there is only one device at the load/unload area. This area provides two functions: the loading of newly arrived samples and the unloading of completed samples. The current system logic at this area attempts to assure that a newly arrived sample never has to wait for a sample holder. Thus, as a sample holder on the loop approaches the Enter point for this area, the system checks to see whether the holder is empty or if it contains a sample that has completed its sequence. If the check satisfies either of these conditions, the system then checks to see if there is room for the sample holder in the load/unload area. This area has room for five sample holders, not including the sample holder on the load/unload device or any holders waiting to merge back onto the loop. If there is room, the sample holder enters the area. A schematic for the load/unload area is shown in Figure 11.3.

As long as there are sample holders in front of the load/unload device, it will continue to operate or cycle. It only stops or pauses if there are no available sample holders to process. The specific action of this device depends on the status of the sample holder and the availability of a new sample. There are four possible actions.

- The sample holder is empty and the new sample queue is empty. In this case, there is no action required, and the sample holder is sent back to the loop.



**Figure 11.2** Test cell schematic.

**Figure 11.3** Load/unload cell schematic.

- The sample holder is empty and a new sample is available. In this case, the new sample is loaded onto the sample holder and sent to the loop.
- The sample holder contains a completed sample, and the new sample queue is empty. In this case, the completed sample is unloaded, and the empty sample holder is sent back to the system.
- The sample holder contains a completed sample, and a new sample is available. In this case, the completed sample is unloaded, and the new sample is loaded onto the sample holder and sent to the loop.

The time for the device to cycle depends on many different factors, but our staff has performed an analysis and concluded that the cycle time follows a triangular distribution with parameters 0.18, 0.23, and 0.45 (minutes). A sample is not considered complete until it is unloaded from its sample holder. At that time, the system will collect the results from its database and forward them to the individual or area requesting the test.

The time for an individual test is constant but depends on the testing cell. These cycle times are given in Table 11.1.

Each test performed at Tester 3 requires 1.6 ounces of reagent, and 38% of the tests at Tester 4 require 0.6 ounces of a different reagent. These are standard reagents and are fed to the testers automatically. Testers periodically fail or require cleaning. Our staff has collected data on these activities, which are given in Table 11.2 for four of the testers. The units for mean time between failures (MTBF) are hours, and the units for mean time to repair (MTR) are minutes.

The testers for Test Cell 2 rarely fail, but they do require cleaning after performing 300 tests. The clean time follows a triangular distribution with parameters (5, 6, 10) (minutes).

The next pilot-testing laboratory will be open 24 hours a day, 7 days a week. Our staff has projected demand data for the site, which are provided in Table 11.3. Hour 1 represents

**TABLE 11.1** Tester Cycle Times in Minutes

Tester	Cycle Time (Minutes)
1	0.77
2	0.85
3	1.03
4	1.24
5	1.7

**TABLE 11.2 Tester Failure and Repair Times**

Tester	MTBF MUT (Hours)	MTR (Minutes)
1	14	11
3	9	7
4	15	14
5	16	13

**TABLE 11.3 Sample Arrival Rates**

Hour	Rate	Hour	Rate	Hour	Rate
1	119	9	131	17	134
2	107	10	152	18	147
3	100	11	171	19	165
4	113	12	191	20	155
5	123	13	200	21	149
6	116	14	178	22	134
7	107	15	171	23	119
8	121	16	152	24	116

the time between midnight and 1 a.m. Hour 24 represents the time between 11 p.m. and midnight. The rate is expressed in average arrivals per hour. The samples arrive without interruption throughout the day at these rates.

Each arriving sample requires a specific sequence of tests, always in the order listed. There are nine possible test sequences with the data given in Table 11.4.

Our contracts generally require that we provide test results within 1 hour from receipt of the sample. For this pilot, we also need to accommodate *Rush* samples for which we must provide test results within 30 minutes. It is estimated that 7% of incoming samples will be labeled Rush. These Rush samples are given preference at the load area.

We requested and received cost figures from the equipment manufacturers. These costs include initial capital, operating, and maintenance costs for the projected life of each unit. The costs given in Table 11.5 are per month per unit.

From this simulation study, we would like to know what configuration would provide the most cost-effective solution while achieving high customer satisfaction. Ideally, we would

**TABLE 11.4 Test Sequences**

Sequence #	Sequence Steps	Percentage (%)
1	1-2-4-5	9
2	3-4-5	13
3	1-2-3-4	15
4	4-3-2	12
5	2-5-1	7
6	4-5-2-3	11
7	1-5-3-4	14
8	5-3-1	6
9	2-4-5	13

**TABLE 11.5 Test Sequences**

Equipment	Cost \$ per Month
Tester Type 1	10,000
Tester Type 2	12,400
Tester Type 3	8,500
Tester Type 4	9,800
Tester Type 5	11,200
Sample holder	387

always like to provide results in less time than the contract requires. However, we also do not feel that the system should include extra equipment just to handle the rare occurrence of a late report.

During a recent SM Testing meeting, a report was presented on the observations of a previous pilot system. The report indicated that completed samples had difficulty entering the load/unload area when the system was loaded lightly. This often caused the completed samples to make numerous loops before they were finally able to exit. A concern was raised that longer-than-necessary test times potentially might cause a system to be configured with excess equipment.

With this in mind, we have approached our equipment vendor and have requested a quote to implement alternate logic at the exit point for the load/unload area only. The proposal gives priority to completed samples exiting the loop. When a sample holder on the loop reaches the Enter point for this area, the system checks the holder to see whether it is empty or contains a sample that has completed its sequence. If the sample holder contains a completed sample and there is room at Load/Unload, it leaves the loop and enters the area. If the sample holder is empty, it checks to see how many sample holders are waiting in the area. If that number is fewer than some suggested number, say 2, the sample holder leaves the loop and enters the area. Otherwise, it continues around the loop. The idea is always to attempt to keep a sample holder available for a new sample, but not to fill the area with empty sample holders. The equipment vendor has agreed to provide this new logic at a one-time cost of \$85,000. As part of your proposal, we would like you to evaluate this new logic, including determining the best value of the suggested number.

Your report should include a recommendation for the most cost-effective system configuration. This should include the number of testers at each cell, the number of sample holders, and a decision on the proposed logic. Please provide all cost estimates on a per-month basis.

We are currently proceeding with the construction of this new facility and will not require a solution until 2 months from now. Since there are several groups competing for this contract, we have decided that we will not provide additional information during the analysis period. However, you are encouraged to make additional, reasonable, documented assumptions. We look forward to receive your report on time and review your proposed solution.

### 11.3 ANSWERING THE BASIC MODELING QUESTIONS

From your reading of the contest problem, you should have some initial ideas for how to approach this modeling effort. The following sections will walk through the simulation modeling steps in order to develop a detailed solution to the contest problem. As you proceed through the sections, you might want to jot down your own ideas and attempt some of

your own analysis. While your approach may be different from what is presented here, your effort and engagement in the problem will be important to how much you get out of this process. Let us begin the modeling effort with a quick iteration through the basic modeling questions.

### ■ What is the system? What information is known by the system?

The purpose of the system is to process test samples within test cells carried by sample holders on a conveyor. The system is well described by the contest problem. Thus, there is no need to repeat the system description here. In summary, the information known by the system is as follows:

- Conveyor speed, total length, and spacing around the conveyor of entry and exit points for the test cells and the load/unload area.
- Load/Unload machine cycle time, TRIA(0.18, 0.23, 0.45) minutes.
- Manual preparation time.
- Cycle times for each tester. See Table 11.1.
- Time to failure and repair distributions for testers 1 and 3–5. See Table 11.2
- Test 2 usage count to cleaning and cleaning time distribution, TRIA(5.0, 6.0, 10.0) minutes.
- Sample arrival mean arrival rate by hour of the day. See Table 11.3.
- Nine different test sequences for the samples along with the probability associated with each sequence.
- A distribution governing the probability of rush samples within the system and a criteria for rush samples to meet (30 minute testing time).
- Equipment costs for testers and sample holders. In addition, the cost of implementing additional logic within the model.

From this initial review of the information and the contest problem description, you should be able to develop an initial list of the Arena™ modeling constructs that may be required in the model.

The initial list should concentrate on identifying primarily data modules. This will help in organizing the data related to the problem. From the list, you can begin to plan (even at this initial modeling stage) on making the model more data driven. That is, by thinking about the input parameters in a more general sense (e.g., expressions and variables), you can build the model in a manner that can improve its usability during experimentation. Based on the current information, a basic list of Arena™ modeling constructs might contain the following:

- CONVEYOR and SEGMENT Modules: To model the conveyor.
- STATION Modules: To model the entry and exit points on the conveyor for the testers and the load/unload cell.
- SEQUENCE Module: To model the test sequences for the samples.
- EXPRESSION Module: To hold the load/unload cycle time and for an array of the cycle times for each tester. Can also be used to hold an expression for the manual preparation time, the sequence probability distribution, and the rush sample distribution.

- FAILURE Modules: To model both count-based (for Tester 2) and time-based failures.
- ARRIVAL Schedule Module: To model the non-stationary arrival pattern for the samples.
- VARIABLE Module: To include various system-wide information such as cost information, number of sample holders, load/unload machine buffer capacity, test cell buffer capacity, and test result criteria.

Once you have some organized thoughts about the data associated with the problem, you can proceed with the identification of the performance measures that will be the major focus of the modeling effort.

### ■ What are the required performance measures?

From the contest problem description, the monthly cost of the system should be considered a performance measure for the system. The cost of the system will certainly affect the operational performance of the system. In terms of meeting the customer's expectations, it appears that the system time for a sample is important. In addition, the contract specifies that test results need to be available within 60 minutes for regular samples and within 30 minutes for rush samples. While late reports do not have to be prevented, they should be rare. Thus, the major performance measures are

- Monthly cost
- Average system time
- Probability of meeting contract system time criteria.

In addition to these primary performance measures, it would be useful to keep track of resource utilization, queue times, size of queues, etc. all of which are natural outputs of a typical Arena™ model.

### ■ What are the entities? What information must be recorded or remembered for each entity? How should the entities be introduced into the system?

Going back to the definition of an entity (an object of interest in the system whose movement or operation within the system may cause the occurrence of events), there appear to be two natural candidates for entities within the model:

- Samples—Arrive according to a pattern and travel through the system
- Sample Holders—Move through the system (e.g., on the conveyor).

To further solidify your understanding of these candidate entities, you should consider their possible attributes. If attributes can be identified, then this should give confidence that these are entities. From the problem, every sample must have a priority (rush or not), an arrival time (to compute total system time), and a sequence. Thus, priority, arrival time, and sequence appear to be natural attributes for a sample. Since a sample holder needs to know whether or not it has a sample, an attribute should be considered to keep track of this for each sample holder. Because both samples and sample holders have clear attributes, there is no reason to drop them from the list of candidate entities.

Thinking about how the candidate entities can enter the model will also assist in helping to understand their modeling. Since the samples arrive according to a non-stationary arrival process (with mean rates that vary by hour of the day), the use of a CREATE module with an

ARRIVAL schedule appears to be appropriate. The introduction of sample holders is more problematic. Sample holders do not arrive to the system. They are just in the system when it starts operating. Thus, it is not immediately clear how to introduce the sample holders. Ask yourself, how can a CREATE module be used to introduce sample holders so that they are always in the system? If all the sample holders arrive at time 0.0, then they will be available when the system starts up. Thus, a CREATE module can be used to create sample holders.

At this point in the modeling effort, a first iteration on entity modeling has been accomplished; however, it is important to understand that this is just a first attempt at modeling and to remember to be open to future revisions. As you go through the rest of the modeling effort, you should be prepared to revise your current conceptual model as deeper understanding is obtained.

To build on the entity modeling, it is natural to ask what resources are used by the entities. This is the next modeling question.

**■ What are the resources that are used by the entities? Which entities use which resources and how?**

It is useful to reconsider the definition of a resource:

- *Resource* A limited quantity of items that are used (e.g., seized and released) by entities as they proceed through the system. A resource has a capacity that governs the total quantity of items that may be available. All the items in the resource are homogeneous, meaning that they are indistinguishable. If an entity attempts to use a resource that does not have units available, it must wait in a queue.

Thus, the natural place to look for resources is where a queue of entities may form. Sample holders (with a sample) wait for testers. In addition sample holders, with or without a sample, wait for the load/unload machine. Thus, the testers and the load/unload machine are natural candidates for resources within the model. In addition, the problem states that:

Samples arriving at the laboratory initially undergo a manual preparation for the automated system. The sample is then placed in the input queue to the load/unload area. This manual preparation typically requires about 5 minutes. When the sample reaches the front of the queue, it waits until an empty sample holder is available.

It certainly appears from this wording that the availability of sample holders can constrain the movement of samples within the model. Thus, sample holders are a candidate for resource modeling.

A couple of remarks about this sort of modeling are probably in order. First, it should be more evident from the things identified as waiting in the queues that samples and sample holders are even more likely to be entities. For example, sample holders wait in queue to get on the conveyor. Now, since sample holders wait to get on a conveyor, does that mean that the conveyor is a candidate resource? Absolutely, yes! In this modeling, you are identifying resources with a little “r.” You are not identifying RESOURCES (also known as things to put in the RESOURCE module)! Be aware that there are many ways to model resources within a simulation (e.g., inventory as a resource with WAIT/SIGNAL). The RESOURCE module is just one very specific method. Just because you identify something as a potential resource, it does not mean you have to use the RESOURCE module to model how it constrains the flow of entities.

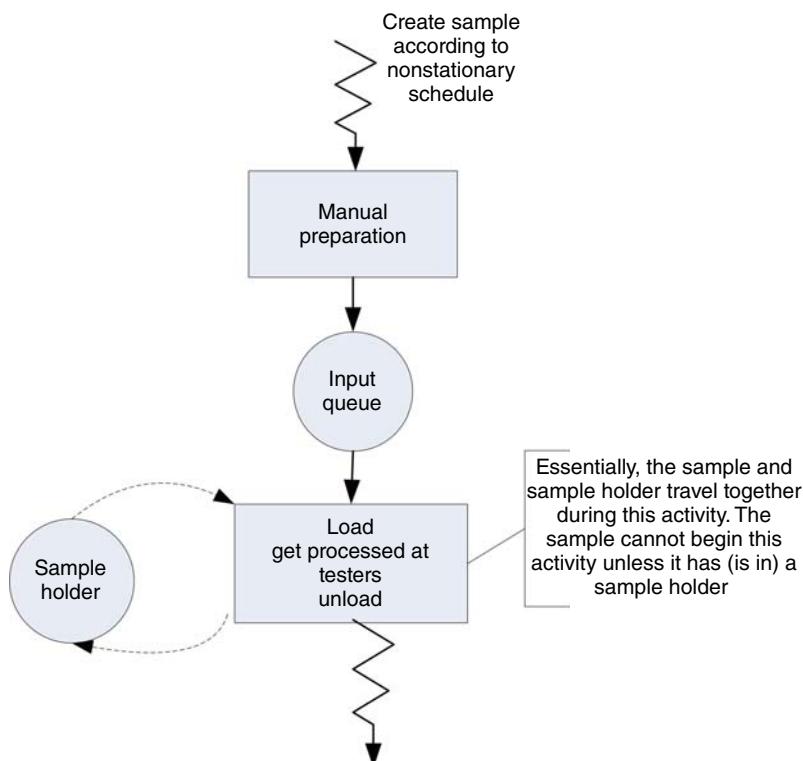
The next step in modeling is to try to better understand the flow of entities by attempting to give a process description.

■ **What are the process flows? Sketch the process or make an activity flow diagram**

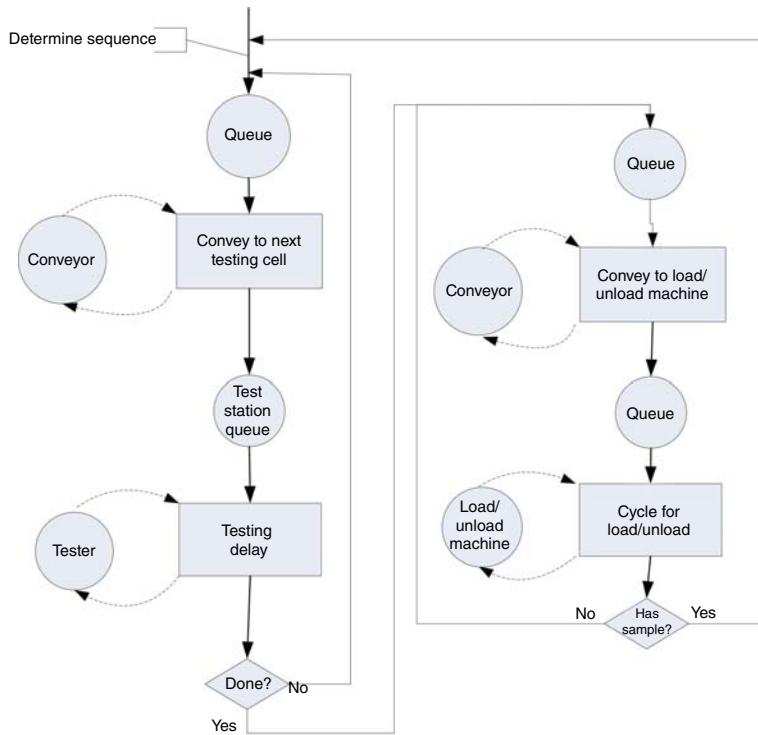
The first thing to remember when addressing process flow modeling is that you are building a *conceptual* model for the process flow. As you should recall, one way to do this is to consider an activity diagram (or some sort of augmented flowchart). Although, in many of the previous modeling examples, there was almost a direct mapping from the conceptual model to the Arena™ model, you should not expect this to occur for every modeling situation. In fact, you should try to build up your conceptual understanding independent of the modeling language. In addition, the level of detail included in the conceptual modeling is entirely up to you! Thus, if you do not know how to model a particular detail then you can just “black box it” or just omit it to be handled later.

Suppose an entity goes into an area for which a lot of detail is required. Just put a box on your diagram and indicate in a general manner what should happen in the box. If necessary, you can come back to it later. If you have complex decision logic that would really clutter up the diagram, then omit it in favor of first understanding the big picture. The complicated control logic for sample holders accessing the load/unload device and being combined with samples is an excellent candidate for this technique.

Before proceeding, you might want to try to sketch out activity diagrams for the samples and the sample holders. Figure 11.4 presents a simplified activity diagram for the samples.



**Figure 11.4** Activity diagram for samples.



**Figure 11.5** Activity cycle diagram for sample holders.

They are created, go through a manual preparation activity, and then flow into the input queue. Whether they wait in the input queue depends upon the availability of the sample holder. Once they have a sample holder, they proceed through the system. The sample must have a sample holder to move through the system. Figure 11.5 illustrates a high level activity cycle diagram for the sample holders. Based on a sequence, they convey (with the sample) to the next appropriate tester. After each test is completed, the sample and holder are conveyed to the next tester in the sequence until they have completed the sequence. At that time, the sample holder and sample are conveyed to the load/unload machine where they experience the load/unload cycle time once they have the load/unload machine. If a sample is not available, the sample holder is conveyed back to the load/unload area.

One thing to notice from this diagram is that the sequence does not have to be determined until the sample and holder are being conveyed to the first appropriate test cell. Also, it is apparent from the diagram that the load/unload machine is the final location visited by the sample and sample holder. Thus, the enter load/unload station can be used as the last location when using the SEQUENCE module. This diagram does not contain the extra logic for testing if the queue in front of a tester is full and the logic associated with the buffer at the load/unload machine.

## 11.4 DETAILED MODELING

Given your conceptual understanding of the problem, you should now be ready to do more detailed modeling in order to prepare for implementing the model within Arena<sup>TM</sup>. When

doing detailed modeling, it is often useful to segment the problem into components that can be more easily addressed. This is the natural problem-solving technique called divide and conquer. Here you must think of the major system components, tasks, or modeling issues that need to be addressed and then work on them first separately and then concurrently in order to have the solution eventually come together as a whole. The key modeling issues for the problem appear to be:

- Conveyor and station modeling (i.e., the physical modeling)
- Modeling samples
- Test Cell modeling including failures
- Modeling sample holders
- Modeling the load/unload area
- Performance measure modeling (cost and statistical collection)
- Simulation horizon and run parameters.

The following sections will examine each of these issues in turn.

#### **11.4.1 Conveyor and Station Modeling**

Recall that conveyor constructs require that each segment of a conveyor be associated with stations within the model. From the problem, the conveyor should be 48 total feet in length. In addition, the problem gives 5 and 3 feet distances between the respective enter and exit points. Since the samples access the conveyor 3 feet from the location that they exited the conveyor, a segment is required for this 3 feet distance for the load/unload area and for each of the test cells. Thus, a station will be required for each exit and enter point on the conveyor. Therefore, there are 12 total stations required in the model. The Exit point for the load/unload machine can be arbitrarily selected as the first station on the loop conveyor.

Figure 11.6 shows the segments for the conveyor. The test cells as well as the load/unload machine have exit and enter stations that define the segments. The exit point is for exiting the cell to access the conveyor. The enter point is for entering the cell (getting off of the conveyor). Since this is a loop conveyor, the last station listed on the segments should be the same as the first station used to start the segments. The problem also indicates that the sample holder takes up 1 foot when riding on the conveyor. Thus, it seems natural to model the cell size for the conveyor as 1 foot.

The conveyor module for this situation is shown in Figure 11.7. The velocity of the conveyor is 1 foot per second with a cell size of 1 foot. Since a sample holder takes up 1 foot on the conveyor and the cell size is 1 foot, the maximum number of cells occupied by an entity on the conveyor is simply 1 cell.

Since the stations are defined, the sequences within the model can now be defined. How you implement the sequences depends on how the conveyor is modeled and on how the physical locations of the work cells are mapped to the concept of stations. As in any model, there are a number of different methods to achieve the same objective. With stations defined for each entry and exit point, the sequences may use each of the entry and exit points. It should be clear that the entry points must be on the sequences; however, because the exit points are also stations, you have the option of using them on the sequences as well. By using the exit point on the sequence, a ROUTE module can be used with the sequence

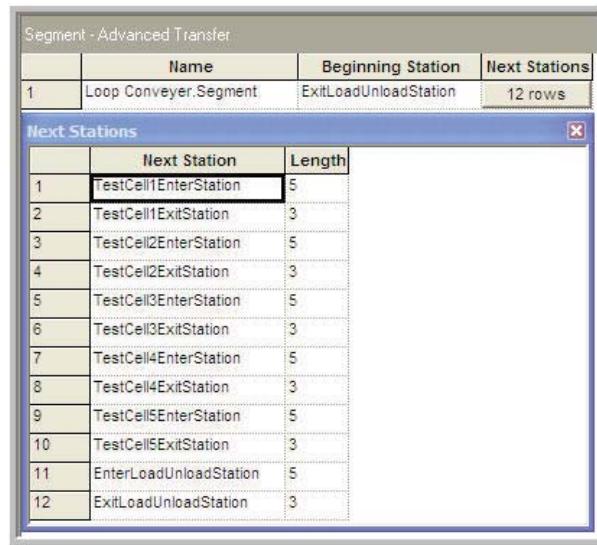


Figure 11.6 Conveyor segments.

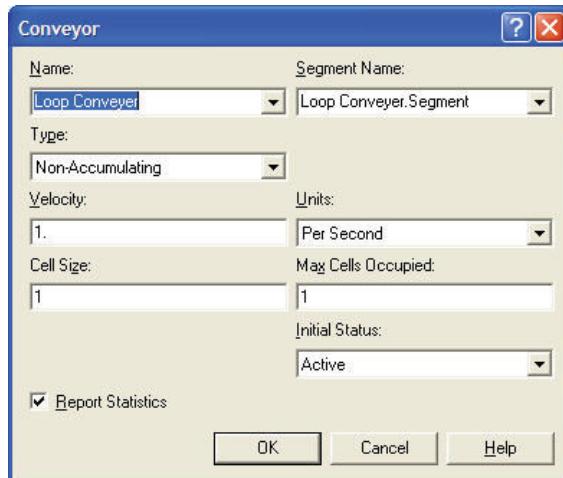


Figure 11.7 Conveyor module.

option to send a sample/sample holder to the appropriate exit station after processing. This could also be achieved by a direct connection, in which case it would be unnecessary to have the exit stations within the sequences. Figure 11.8 illustrates the sequence of stations for test sequence 1 for the problem. Notice that the stations alternate (enter then exit) and that the last station is the station representing the entry point for the load/unload area.

In order to randomly assign a sequence, the sequences can be placed into a set and then the index into the set randomly generated via the appropriate probability distribution. Figure 11.9 shows the Advanced Set module used to define the set of sequences for

Sequence - Advanced Transfer			Steps				
	Name	Steps		Station Name	Step Name	Next Step	Assignments
1	Sequence1	9 rows	1	TestCell1EnterStation			0 rows
2	Sequence2	7 rows	2	TestCell1ExitStation			0 rows
3	Sequence3	9 rows	3	TestCell2EnterStation			0 rows
4	Sequence4	7 rows	4	TestCell2ExitStation			0 rows
5	Sequence5	7 rows	5	TestCell4EnterStation			0 rows
6	Sequence6	9 rows	6	TestCell4ExitStation			0 rows
7	Sequence7	9 rows	7	TestCell5EnterStation			0 rows
8	Sequence8	7 rows	8	TestCell5ExitStation			0 rows
9	Sequence9	7 rows	9	EnterLoadUnloadStation			0 rows

Figure 11.8 Test Sequence 1.

Advanced Set - Advanced Process				Members	
	Name	Set Type	Members		Other
1	TesterQueueSet	Queue	5 rows	1	Sequence1
2	EnterTestingStationSet	Other	5 rows	2	Sequence2
3	ExitTestingStationSet	Other	5 rows	3	Sequence3
4	ConveyorQueueSet	Queue	5 rows	4	Sequence4
5	SequenceSet	Other	9 rows	5	Sequence5
Double-click here to add a new row.				6	Sequence6
				7	Sequence7
				8	Sequence8
				9	Sequence9

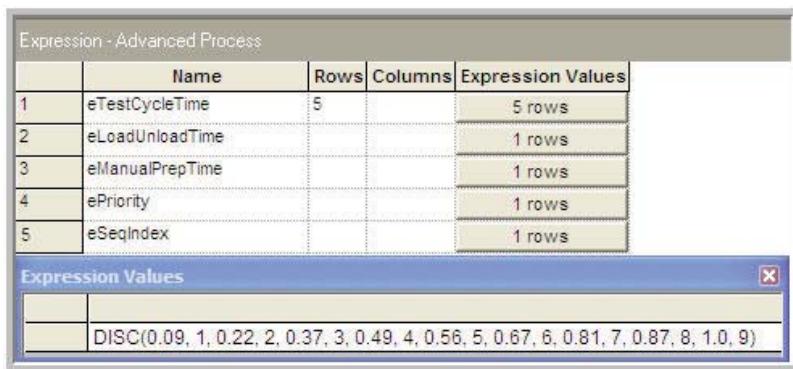
Figure 11.9 Set for holding sequences.

the model. In addition, Figure 11.10 shows the implementation of the distribution across the sequences as an expression, called *eSeqIndex*, using the DISC() function within the EXPRESSION module.

#### 11.4.2 Modeling Samples and the Test Cells

When modeling a large complex problem, you should try to start with a simplified situation. This allows a working model to be developed without unnecessarily complicating the modeling effort. Then, as confidence in the basic model improves, enhancements to the basic model can take place. It will be useful to do just this when addressing the modeling of samples and sample holders.

As indicated in the activity diagrams, once a sample has a sample holder, the sample holder and sample essentially become one entity as they move through the system. Thus, a sample also follows the basic flow as laid out in Figure 11.5. This is essentially what happens to the sample when it is in the black box of Figure 11.4. Thus, to simplify the modeling,



**Figure 11.10** Discrete distribution for assigning random sequence.

it is useful to assume that a sample holder is always available whenever a sample arrives. Since sample holders are not being modeled, there is no need to model the details of the load/unload machine. This assumption implies that once a sample completes its sequence, it can simply exit at the load/unload area and that any newly arriving samples simply get on at the load/unload area. With these assumptions, the modeling of the sample holders and the load/unload stations (including its complicated rules) can be bypassed for the time being.

From the conceptual model (activity diagram), it should be clear that the testers can be easily modeled with the RESOURCE module. In addition, the diagram indicates that the logic at any test cell is essentially the same. This could lead to the use of generic stations. Based on all these ideas, you should be able to develop pseudo-code for this situation. If you are following along, you might want to pause and sketch out your own pseudo-code for the simplified modeling situation.

Exhibit 11.1 shows possible pseudo-code for this initial modeling. Samples are created according to a non-stationary arrival pattern and then are routed to the exit point for the load/unload station. Then, the samples access the conveyor and are conveyed to the appropriate test cell according to their sequence. Once at a test cell, they test if there is room to get off the conveyor. If so, they exit the conveyor and then use the appropriate tester (SEIZE, DELAY, RELEASE). After using the tester, they are routed to the exit point for the test cell, where they access the conveyor and are conveyed to the next appropriate station. If space is not available at the test cell, they do not exit the conveyor, but rather are conveyed back to the test cell to try again. Once the sample has completed its sequence, the sample will be conveyed to enter load/unload area, where it exits the conveyor and is disposed.

Given the pseudo-code and the previous modeling, you should be able to develop an initial Arena™ model for this simplified situation. For practice, you might try to implement the ideas that have been discussed before proceeding. The Arena™ model (with animation) representing this initial modeling is given in the file *SMTTestingInitialModeling.doe*. The flowchart modules corresponding to the pseudo-code are shown in Figure 11.11. The following approach was taken when developing the initial model for samples and test cells:

---

**Exhibit 11.1** Pseudo-Code Sample Process

---

```

CREATE sample according to non-stationary pattern
ASSIGN
    myEnterSystemTime = TNOW
    myPriorityType ~ Priority CDF
END ASSIGN
DELAY for manual preparation time
ROUTE to ExitLoadUnloadStation

STATION ExitLoadUnLoadStation
ASSIGN
    mySeqIndex ~ Sequence CDF
    Entity.Sequence = MEMBER(SequenceSet, mySeqIndex)
    Entity.JobStep = 0
END ASSIGN
ACCESS Loop Conveyor
CONVEY by Sequence

STATION Generic Testing Cell Enter
DECIDE
    IF NQ at testing cell < cell waiting capacity THEN
        EXIT Loop Conveyor
        SEIZE 1 unit of tester
        DELAY for testing time
        RELEASE 1 unit of tester
        ROUTE to Generic Testing Cell Exit
    ELSE
        CONVEY back to Generic Testing Cell Enter
    END IF
END DECIDE

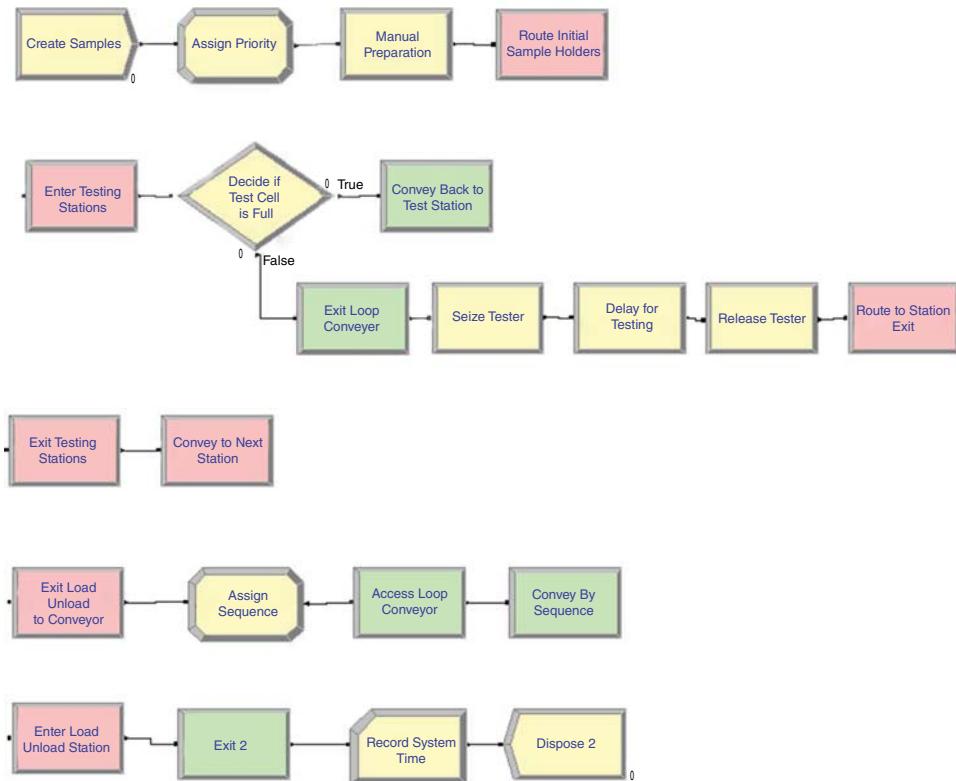
STATION Generic Testing Cell Exit
ACCESS Loop Conveyor
CONVEY by Sequence

STATION EnterLoadUnloadStation
EXIT Loop Conveyor
RECORD System time
DISPOSE

```

---

- *Variables.* A variable array, vTestCellCapacity(5), was defined to hold the capacity of each test cell. While for this particular problem, the cell capacity was the same for each cell by making a variable array; the cell capacity can be easily varied if needed when testing the various design configurations.
- *Expressions.* An arrayed expression, eTestCycleTime(5), was defined to hold the cycle times for the testing machines. By making these expressions, they can be easily changed from one location in the model. In addition, expressions were defined for the manual preparation time (eManualPrepTime), the priority distribution (ePriority), and the sequence distribution (eSeqIndex).
- *Resources.* Five separate resources were defined to represent the testers (Cell1Tester, Cell2Tester, Cell3Tester, Cell4Tester, Cell5Tester)
- *Sets.* A resource set, TestCellResourceSet, was defined to hold the test cell resources for use in generic modeling. An entity picture set (SamplePictSet) with nine members



**Figure 11.11** Initial model for samples and test cells.

was defined to be able to change the picture of the sample according to the sequence it was following. A queue set, TesterQueueSet, was defined to hold the queues in front of each tester for use in generic modeling. In addition, a queue set, ConveyorQueueSet, was defined to hold the access queue for the conveyor at each test cell. Two station sets were defined to hold the enter (EnterTestingStationSet) and the exit (ExitTestingStationSet) stations for generic station modeling. Finally, a sequence set, SequenceSet, was used to hold the nine sequences followed by the samples.

- *Schedules.* An ARRIVAL schedule (see Figure 11.12) was defined to hold the arrival rates by hour to represent the non-stationary arrival pattern for the samples.
- *Failures.* Five failure modules were used to model the failure and cleaning associated with the test cells. Four failure modules (Tester 1 Failure, Tester 3 Failure, Tester 4 Failure, and Tester 5 Failure) defined time-based failures and the appropriate repair times. A count-based failure was used to model Tester 2's cleaning after 300 uses. The failures are illustrated in Figure 11.13. An important assumption with the use of the FAILURE module is that the entire resource becomes failed when a failure occurs. It is not clear from the contest problem specification what would happen at a test cell that has more than one tester when a failure occurs. Thus, for the sake of simplicity, it will be useful to assume that each unit of a multiple unit tester does not fail individually.

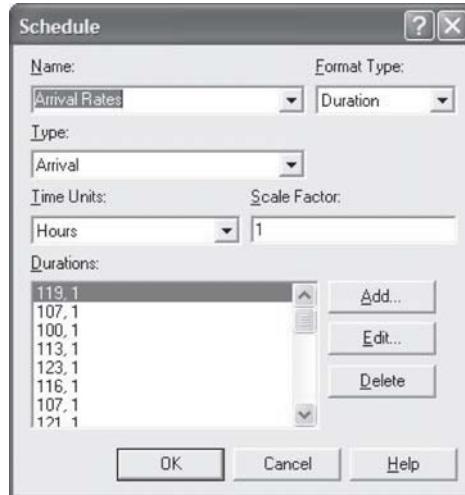


Figure 11.12 ARRIVAL schedule for samples.

Failure - Advanced Process									
	Name	Type	Up Time	Up Time Units	Count	Down Time	Down Time Units	Uptime in this State only	
1	Tester 1 Failure	Time	EXPO(14)	Hours	Hours	EXPO(11)	Minutes		
2	Tester 3 Failure	Time	EXPO(9)	Hours	Hours	EXPO(7)	Minutes		
3	Tester 4 Failure	Time	EXPO(15)	Hours	Hours	EXPO(14)	Minutes		
4	Tester 5 Failure	Time	EXPO(18)	Hours	Hours	EXPO(13)	Minutes		
5	Tester 2 Failure	Count	1.0	Hours	300	TRIA( 5, 6 , 10 )	Minutes		

Figure 11.13 FAILURE modules for SMTTesting.

As a first step towards verifying that the model is working as intended, the initial model, *SMTTestingInitialModeling.doe*, can be modified so that only 1 entity is created. For each of the nine different sequences, the total distance traveled on the sequence and the total time for testing can be calculated. For example, for the first sequence, the distance from load/unload exit to the enter location of cell 1 is 5 feet, the distance from cell 1 exit to cell 2 enter is 5 feet, the distance from cell 2 exit to cell 4 enter is 13 feet, the distance from cell 4 exit to cell 5 enter is 5 feet, and finally, the distance from cell 5 exit to the enter point for load/unload is 5 feet. This totals 33 feet as shown in Table 11.6. The time to travel 33 feet at 1 foot per second is 0.55 minutes. The total testing time for this sequence is  $(0.77 + 0.85 + 1.24 + 1.77 = 5.11)$  minutes. Finally, the preparation time is 5 minutes for all sequences. Thus, the total time that it should take a sample following sequence 1 should be 10.11 minutes assuming no waiting and no failures. To check that the model can reproduce this quantity, nine different model runs were made such that the single entity followed each of the nine different sequences. The total system time for the single entity was recorded and matched exactly those figures given in Table 11.6. This should give you confidence that the current implementation is working correctly, especially that there are no problems with the sequences. The modified model for testing sequence 9 is given in Arena™ file, *SMTTestingInitialModelingTest1Entity.doe*.

**TABLE 11.6 Single Sample System Time**

Sequence #	Sequence Steps	Distance (feet)	Travel Time	Test Time	Preparation Time	Total Time
1	1-2-4-5	33	0.55	5.11	5.0	10.11
2	3-4-5	36	0.60	3.97	5.0	9.57
3	1-2-3-4	33	0.55	3.89	5.0	9.44
4	4-3-2	132	2.20	3.12	5.0	10.32
5	2-5-1	84	1.40	3.32	5.0	9.72
6	4-5-2-3	81	1.35	4.82	5.0	11.17
7	1-5-3-4	81	1.35	4.74	5.0	11.09
8	5-3-1	132	2.20	3.50	5.0	10.70
9	2-4-5	36	0.6	3.79	5.0	9.39

This testing also provides a lower bound on the expected system times for each of the sequences.

Now that a preliminary working model is available, an initial investigation of the resources required by the system and further verification can be accomplished. From the sequences that are given, the total percentage of the samples that will visit each of the test cells can be tabulated. For example, test cell 1 is visited in sequences 1, 3, 5, 7, and 8. Thus, the total percentage of the arrivals that must visit test cell 1 is  $(0.09 + 0.15 + 0.07 + 0.14 + 0.06 = 0.51)$ . The total percentage for each of the test cells is given in Table 11.7.

Using the total percentages given in Table 11.7, the mean arrival rate to each of the test cells for each hour of the day can be computed. From the data given in Table 11.3, you can see that the minimum hourly arrival rate is 100 and that it occurs in the 3rd hour. The maximum hourly arrival rate of 200 customers per hour occurs in the 13th hour. From the minimum and maximum hourly arrival rates, you can get an understanding of the range of resource requirements for this system. This will not only help when solving the problem but will also help in verifying that the model is working as intended.

Let us look at the peak arrival rate first. From the discussion in Chapter 8, the offered load is a dimensionless quantity that gives the average amount of work offered per time unit to the  $c$  servers in a queuing system. The offered load is defined as  $r = \lambda/\mu$ . This can be interpreted as each customer arriving with  $1/\mu$  average units of work to be performed. It can also be thought of as the expected number of busy servers if there are an infinite number of servers. Thus, the offered load can give a good idea of about how many servers might be utilized. Table 11.8 calculates the offered load for each of the test cells under the peak arrival rate.

**TABLE 11.7 Percentage of Arrivals Visiting Each Test Cell**

Cell	Percentage from Sequences					Total Percentage
1	0.09	0.15	0.07	0.14	0.06	0.51
2	0.09	0.15	0.12	0.07	0.11	0.67
3	0.13	0.15	0.12	0.07	0.11	0.13
4	0.09	0.13	0.15	0.12	0.11	0.14
5	0.09	0.13	0.07	0.11	0.14	0.06
						0.13
						0.73

**TABLE 11.8 Offered Load Calculation for Peak Hourly Rate**

Test Cell	Arrival Rate (Per Hour)	Testing Time (Minutes)	Testing Time (Hours)	Service Rate (Per Hour)	Offered Load
1	102	0.77	0.01283	77.92	1.31
2	134	0.85	0.01417	70.59	1.90
3	142	1.03	0.01717	58.25	2.44
4	174	1.24	0.02067	48.39	3.60
5	146	1.70	0.02833	35.29	4.14

Number Busy	Average	Half Width
Cell1Tester	1.3132	0.017101725
Cell2Tester	1.8917	0.018252301
Cell3Tester	2.4236	0.024626714
Cell4Tester	3.5740	0.034768944
Cell5Tester	4.1244	0.047537947

**Figure 11.14** Results from offered load experiment.**TABLE 11.9 Offered Load Calculation for Minimum Hourly Rate**

Test Cell	Arrival Rate (Per Hour)	Testing Time (Minutes)	Testing Time (Hours)	Service Rate (Per Hour)	Offered Load
1	51	0.77	0.01283	77.92	0.65
2	67	0.85	0.01417	70.59	0.95
3	71	1.03	0.01717	58.25	1.22
4	87	1.24	0.02067	48.39	1.80
5	73	1.70	0.02833	35.29	2.07

The arrival rate for the first test cell is  $0.51 \times 200 = 102$ . Thus, you can see that a little over 1 server can be expected to be busy at the first test cell under peak loading conditions.

You can confirm these numbers by modifying the Arena™ model so that the resource capacities of the test cells are infinite and by removing the failure modules from the resources. In addition, the CREATE module will need to be modified so that the arrival rate is 200 samples per hour. If you run the model for ten 24-hour days, the results shown in Figure 11.14 will be produced. As can be seen in the figure, the results match very closely to that of the offered load analysis.

These results provide additional evidence that the initial model is working properly and indicate how many units of each test cell might be needed under peak conditions. The model is given in the file *SMTTestingInitialModelingOnResources.doe*. Table 11.9 indicates the offered load under the minimum arrival rate.

Based on the analysis of the offered load, a preliminary estimate of the resource requirements for each test cell can be determined as in Table 11.10. The requirements were determined by rounding up the computed offered load for each test cell. This provides a

**TABLE 11.10 Offered Load Calculation for Minimum Hourly Rate**

Test Cell	Resource Requirements	
	Low	High
1	1	2
2	1	2
3	2	3
4	2	4
5	3	5

range of values since it is not clear that designing to the maximum is necessary, given the non-stationary behavior of the arrival of samples.

#### 11.4.3 Modeling Sample Holders and the Load/Unload Area

Now that a basic working model has been developed and tested, you should feel comfortable with more detailed modeling involving the sample holders and the load/unload area. From the previous discussion, the sample holders appeared to be an excellent candidate for being modeled as an entity. In addition, it also appeared that the sample holders could be modeled as a resource because they also constrain the flow of the sample if a sample holder is not available. There are a number of modeling approaches possible for addressing this situation. By considering the functionality of the load/unload machine, the situation may become more clear. The problem states that:

As long as there are sample holders in front of the load/unload device, it will continue to operate or cycle. It only stops or pauses if there are no available sample holders to process.

Since the load/unload machine is clearly a resource, the facts that sample holders wait in front of it and that it operates on sample holders indicate that sample holders should be entities. Further consideration of the four possible actions of the load/unload machine can provide some insight on how samples and sample holders interact. As a reminder, the following are the four actions.

1. *The sample holder is empty and the new sample queue is empty.* In this case, there is no action required, and the sample holder is sent back to the loop.
2. *The sample holder is empty and a new sample is available.* In this case, the new sample is loaded onto the sample holder and sent to the loop.
3. *The sample holder contains a completed sample, and the new sample queue is empty.* In this case, the completed sample is unloaded, and the empty sample holder is sent back to the system.
4. *The sample holder contains a completed sample, and a new sample is available.* In this case, the completed sample is unloaded, and the new sample is loaded onto the sample holder and sent to the loop.

Thus, if a sample holder contains a sample, it is unloaded during the load/unload cycle. In addition, if a sample is available, it is loaded onto the sample holder during the load/unload cycle. The cycle time for the load/unload machine varies according to a triangular

distribution, but it appears as if both the loading and/or the unloading can happen during the cycle time. Thus, whether there is a load, a unload, or both, the time using the load/unload machine is triangularly distributed. If the sample holder has a sample, then during the load/unload cycle, they are separated from each other. If a new sample is waiting, then the sample holder and the new sample are combined together during the processing. This indicates two possible approaches to modeling the interaction between sample holders and samples:

- **BATCH and SEPARATE.** The BATCH module can be used to batch the sample and the sample holder together. Then the SEPARATE module can be used to split them apart.
- **PICKUP and DROPOFF.** The PICKUP module can be used to have the sample holder pick up a sample, and the DROPOFF module can be used to have the sample holder drop off a completed sample.

Of the two methods, the PICKUP/DROPOFF approach is potentially easier since it puts the sample holder in charge. In the BATCH/SEPARATE approach, it will be necessary to properly coordinate the movement of the sample and the sample holder (perhaps through MATCH, WAIT, SIGNAL modules). In addition, the BATCH module requires careful thought about how to handle the formation of the representative entity and its attributes. In what follows, the PICKUP/DROPOFF approach will be used. As an exercise, you might attempt the BATCH/SEPARATE approach.

To begin the modeling of sample holders and samples, you need to think about how they are created and introduced into the model. The samples should continue to be created according to the non-stationary arrival pattern, but after preparation occurs, they need to wait until they are picked up by a sample holder. This type of situation can be modeled very effectively with a HOLD module with the infinite hold option specified. In a sense, this is just like the bus system situation of Chapter 10. Now you should be able to sketch out the pseudo-code for this situation.

The pseudo-code for this situation has been developed as shown in Exhibit 11.2. As seen in the exhibit, as a sample holder arrives at the enter point for the load/unload station, it first checks to see if there is space in the load/unload buffer. If not, it is simply conveyed back around. If there is space, it checks to see if it is empty and there are no waiting samples. If so, it can simply convey back around the conveyor. Finally, if it is not empty or if there is a sample waiting, it will exit the conveyor to try to use the load/unload machine. If the machine is busy, the sample holder must wait in a queue until the machine is available. Once the machine is seized, it can drop off the sample (if it has one) and pick up a new sample (if one is available). After releasing the load/unload machine, the sample holder possibly with a picked up sample, goes to the exit load/unload station to try to access the conveyor. If it has a sample, it conveys by sequence; otherwise, it conveys back around to the entry point of the load/unload area.

There is one final issue that needs to be addressed concerning the sample holders. How should the sample holders be introduced into the model? As mentioned previously, a CREATE module can be used to create the required number of sample holders at time 0.0. Then, the newly created sample holders can be immediately sent to the *ExitLoadUnloadStation*. This will cause the newly created and empty sample holders to try to access the conveyor. They will ride around on the empty conveyor until samples start to arrive. At which time, the logic will cause them to exit the conveyor to pick up samples.

---

**Exhibit 11.2 Pseudo-Code Load/Unload Cell**

---

```

CREATE sample according to non-stationary pattern
ASSIGN
    myEnterSystemTime = TNOW
    myPriorityType ~ Priority CDF
END ASSIGN
DELAY for manual preparation time
HOLD until picked up by sample holder

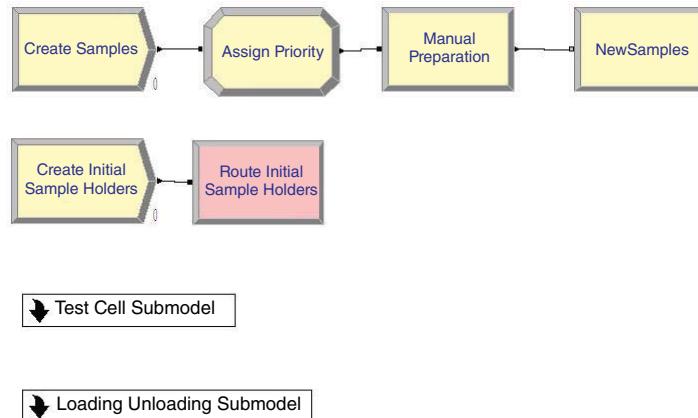
STATION EnterLoadUnloadStation
DECIDE
    IF NQ at load/unload cell ≥ buffer capacity THEN
        CONVEY back to EnterLoadUnloadStation
    ELSE
        IF sample holder is empty and new sample queue is empty THEN
            CONVEY back to EnterLoadUnloadStation
        ELSE
            EXIT Loop Conveyor
            SEIZE load/unload machine
            DROPOFF sample if loaded
            DELAY for load/unload cycle time
            IF n THEN new sample queue is not empty
                PICKUP new sample
            END IF
            RELEASE load/unload machine
            ROUTE to ExitLoadUnLoadStation
        END IF
    END IF
END DECIDE

STATION ExitLoadUnLoadStation
DECIDE
    IF the sample holder has a sample THEN
        ASSIGN mySeqIndex ~ Sequence CDF
        Entity.Sequence = MEMBER(SequenceSet, mySeqIndex)
        Entity.JobStep = 0
    END IF
END DECIDE
ACCESS Loop Conveyor
DECIDE
    IF the sample holder has a sample THEN
        CONVEY by Sequence
    ELSE
        CONVEY to EnterLoadUnloadStation
    END IF
END DECIDE

```

---

Figure 11.15 provides an overview of the entire model. The samples are created, have their priority assigned, experience manual preparation, and then wait in an infinite hold until picked up by the sample holders. The hold queue for the samples is ranked by the priority of the sample, in order to give preference to rush samples. The initial sample holders are created at time 0.0 and routed to the *ExitLoadUnloadStation*. Two Arena™ submodels were used to model the test cells and the load/unload area. The logic for the test cell submodel is exactly the same as described during the initial model development.



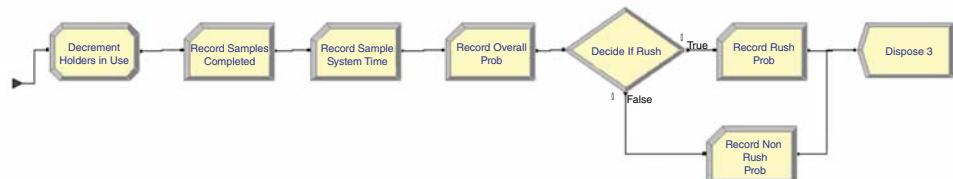
**Figure 11.15** Overview of Entire Model.

The file *SMTTesting.doe* contains the completed model. The logic described in Exhibit 11.2 is implemented in the submodel, Loading Unloading Submodel. The modeling of the alternate logic is also included in this model.

Now that the basic model is completed, you can concentrate on issues related to running and using the results of the model.

#### 11.4.4 Performance Measure Modeling

Of the modeling issues identified at the beginning of this section, the only remaining issues are the collection of the performance statistics and the simulation run parameters. The key performance statistics are related to the system time and the probability of meeting the contract time requirements. These statistics can be readily captured using RECORD modules. The submodel for the load/unload area has an additional submodel that implements the collection of these statistics using RECORD modules. Figure 11.16 illustrates this portion of the model. A variable that keeps track of the number of sample holders in use is decremented each time a sample holder drops off a sample (and incremented any time a sample holder picks up a sample). After the sample is dropped off, the number of samples completed is recorded with a counter and the system time is tallied with a RECORD module. The overall probability of meeting the 60-minute limit is tallied using a Boolean expression. Then, the probability is recorded according to whether or not the sample was a rush or not.



**Figure 11.16** System time statistical collection.

Statistic - Advanced Process				
	Name	Type	Expression	Report Label
1	Tester1Cost	Output	MR(Cell1Tester) * vTesterCostPerMonth ( 1 )	Tester1Cost
2	Tester2Cost	Output	MR(Cell2Tester) * vTesterCostPerMonth ( 2 )	Tester2Cost
3	Tester3Cost	Output	MR(Cell3Tester) * vTesterCostPerMonth ( 3 )	Tester3Cost
4	Tester4Cost	Output	MR(Cell4Tester) * vTesterCostPerMonth ( 4 )	Tester4Cost
5	Tester5Cost	Output	MR(Cell5Tester) * vTesterCostPerMonth ( 5 )	Tester5Cost
6	TotTesterCost	Output	OVALUE(Tester1Cost) + OVALUE(Tester2Cost) + OVALUE(Tester3Cost) + OVALUE(Tester4Cost) + OVALUE(Tester5Cost)	TotTesterCost
7	HolderCost	Output	vNumHolders * vCostPerHolderPerMonth	HolderCost
8	LogicCost	Output	vNewLogicCost * vNewLogicOption	LogicCost
9	TotalCost	Output	OVALUE(HolderCost)+OVALUE(TotTesterCost) + OVALUE(LogicCost)	TotalCost

Figure 11.17 Cost collection with output statistics.

TABLE 11.11 High Resource Case Cost Example

Equipment	Cost per Month (\$)	# Units	Total Cost (\$)
Tester Type 1	10,000	2	20,000
Tester Type 2	12,400	2	24,800
Tester Type 3	8,500	3	25,500
Tester Type 4	9,800	4	39,200
Tester Type 5	11,200	5	56,000
Sample Holder	387	16	6,192
		Total	171,692

The cost of each configuration can also be tabulated using output statistics as shown in Figure 11.17. Even though there is no randomness in these cost values, these values can be captured to facilitate the use of the Process Analyzer and OptQuest.

For example, Table 11.11 shows the total monthly cost calculation for the high resource case assuming the use of 16 sample holders.

Now that the model has been set up to collect the appropriate statistically quantities, the simulation time horizon and run parameters must be determined for the experiments.

#### 11.4.5 Simulation Horizon and Run Parameters

The setting of the simulation horizon and the run parameters is challenging within this problem because of lack of guidance on this point from the problem description. Without specific guidance, you will have to infer from the problem description how to proceed.

It can be inferred from the problem description that SMTTesting is interested in using the new design over a period of time, which may be many months or years. In addition, the problem states that SMTTesting wants all cost estimates on a per month basis. In addition, the problem provides that during each day the arrival rate varies by hour of the day, but in the absence of any other information, it appears that each day repeats this same non-stationary behavior. Because of the non-stationary behavior during the day, steady-state analysis is inappropriate based solely on data collected *during* a day. However, since each day repeats, the *daily* performance of the system may approach steady state over many days.

If the system starts empty and idle on the first day, the performance from the initial days of the simulation may be affected by the initial conditions. This type of situation has been termed steady-state cyclical parameter estimation by Law [2007] and is a special case of steady-state simulation analysis. Thus, this problem requires an investigation of the effect of initialization bias.

To make this more concrete, let  $X_i$  be the average performance from the  $i$ th day. For example, the average system time from the  $i$ th day. Then, each  $X_i$  is an observation in a time series representing the performance of the system for a sequence of days. If the simulation is executed with a run length of, say, 30 days, then the  $X_i$  constitute *within replication* statistical data, as described in Chapter 7. Thus, when considering the effect of initialization bias, the  $X_i$  need to be examined over multiple replications. Therefore, the warm-up period should be in terms of days. The challenge then becomes collecting the daily performance.

There are a number of methods to collect the required daily performance in order to perform an initialization bias analysis. For example, you could create an entity each day and use that entity to record the appropriate performance; however, this would entail special care during the tabulation and would require you to implement the capturing logic for each performance measure of interest. The simplest method is to take advantage of the settings on the Run Setup dialog concerning how the statistics and the system are initialized for each replication.

Let us go over the implication of these initialization options. Since there are two check boxes, one for whether or not the statistics are cleared at the end of the replication and one for whether or not the system is reset at the beginning of each replication, there are four possible combinations to consider. The four options and their implications are as follows:

- *Statistics Checked and System Checked.* This is the default setting. With this option, the statistics are reset (cleared) at the end of each replication and the system is set back to empty and idle. The net effect is that each replication has independent statistics collected on it and each replication starts in the same (empty and idle) state. If a warm-up period is specified, the statistics that are reported are only based on the time after the warm-up period. Up until this point, this option has been used throughout the book.
- *Statistics Unchecked and System Checked.* With this option, the system is reset to empty and idle at the end of each replication so that each replication starts in the same state. Since the statistics are unchecked, the statistics will accumulate across the replications. That is, the average performance for the  $j$ th replication includes the performance for all previous replications and the  $j$ th replication. If you were to write out the statistics for the  $j$ th replication, it would be the cumulative average up to and including that replication. If a warm-up period is specified, the statistics are cleared at the warm-up time and then captured for each replication. Since this clears the accumulation, the net effect of using a warm-up period in this case is that the option functions like option 1.
- *Statistics Checked and System Unchecked.* With this option, the statistics are reset (cleared) at the end of each replication, but the system is not reset. That is, each subsequent replication begins its initial state using the final state from the previous replication. The value of TNOW is not reset to zero at the end of each replication. In this situation, the average performance for each replication does not include the performance from the previous replication; however, they are not independent observations since the state of the system was not reset. If a warm-up period is specified, a

special warm-up summary report appears on the standard text-based Arena™ report at the warm-up time. After the warm up, the simulation is then run for the specified number of replications for each run length. Based on how option 3 works, an analyst can effectively implement their own batch means method based on time-based batch intervals by specifying the replication length as the batching interval. The number of batches is based on the number of replications (after the warm-up period).

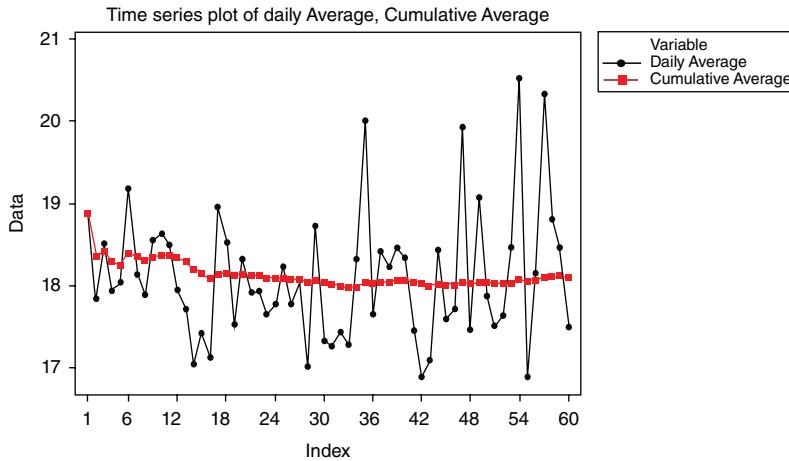
- *Statistics Unchecked and System Unchecked.* With this option, the statistics are not reset and the system state is not reset. The statistics accumulate across the replications and subsequent replications use the ending state from the previous replication. If a warm-up period is specified a special warm-up summary report appears on the standard text-based Arena™ report at the warm-up time. After the warm up, the simulation is then run for the specified number of replications for each run length.

According to these options, the current simulation can be set up with option 3 with each replication lasting 24 hours and use the number of replications to get the desired number of days of simulation. Since the system is not reset, the variable TNOW is not set back to zero. Thus, each replication causes additional time to evolve during the simulation. The effective run length of the simulation will be determined by the number of days specified for the number of replications. For example, a specification of 60 replications will result in 60 days of simulated time. If the statistics are captured to a file for each replication (day), then the performance by day will be available to analyze for initialization bias.

One additional problem in determining the warm-up period is the fact that there will be many design configurations to be examined. Thus, a warm-up period will need to be picked that will work on the range of design configurations that will be considered; otherwise, a warm-up analysis will have to be performed for every design configuration. In what follows, the effect of initialization bias will be examined to try to find a warm-up period that will work well across a wide variety of design configurations.

The model was modified to run the high resource case from Table 11.10 using 48 sample holders. In addition, the model was modified so that the distributions were parameterized by a stream number. This was done so that separate invocations of the program would generate different sample paths. An output statistic was defined to capture the average daily system time for each of the replications and the model was executed 10 different times after adjusting the stream numbers to get 10 independent sample paths of 60 days in length. The 10-sample paths were averaged to produce the Welch plot as shown in Figure 11.18.

From the plot, there is no discernible initialization bias. Thus, it appears that if the system has enough resources, there does not seem to be a need for a warm-up period. However, if the low resource case of Table 11.10 is run under the same conditions, an execution error will occur that indicates that the maximum number of entities for the professional version of Arena™ is exceeded. In this case, the overall arrival rate is too high for the available resources in the model. The samples build up in the queues (especially the input queue) and unprocessed samples will be carried forward each day, eventually building up to a point that exceeds the number of entities permitted. Thus, it should be clear that in the under resourced case, the performance measures cannot reach steady state. Whether or not a warm-up period is set for these cases, they will naturally be excluded as design alternatives because of exceptionally poor performance. Therefore, setting a warm-up period for under-capacitated design configurations appears unnecessary. Based on this analysis, a 1-day warm-up period will be used to be very conservative across all of the design configurations.



**Figure 11.18** The Welch plot for high resource configuration.

The run setup parameter settings for the initialization bias investigation do not necessarily apply to further experiments. In particular, with the third option, the days are technically within replication data. Option 1 appears to be applicable to future experiments. For option 1, the run length and the number of replications for the experiments need to be determined. The batch means method based on one long replication of many days or the method of replication deletion can be utilized in this situation. Because you might want to use the Process Analyzer and possibly OptQuest during the analysis, the method of replication deletion may be more appropriate. Based on the rule of thumb in Banks et al. [2005], the run length should be at least 10 times the amount of data deleted. Because 1 day of data is being deleted, the run length should be set to 11 days so that a total of 10 days of data will be collected. Now, the number of replications needs to be determined.

Using the high resource case with 48 sample holders, a pilot run was made of five replications of 11 days with 1-day warm up. The user-defined results of this run are shown in Figure 11.19.

<b>Tally</b>		
Expression	Average	Half Width
Record Non Rush Prob	0.9983	0.00
Record Overall Prob	0.9984	0.00
Record Rush Prob	0.9600	0.00
SampleSystemTime	17.8112	0.37

<b>Counter</b>		
Count	Average	Half Width
SamplesCompleted	33688.00	150.40

**Figure 11.19** User-defined results for pilot run.

As indicated in the figure, there were a large number of samples completed during each 10-day period and the half-widths of the performance measures are already very acceptable with only five replications. Thus, five replications will be used in all the experiments that follow. Because other configurations may have more variability and thus require more than five replications, a risk is being taken that adequate decisions will be able to be made across all design configurations. Thus, a trade-off is being made between the additional time that pilot runs would entail and the risks associated with making a valid decision.

#### **11.4.6 Preliminary Experimental Analysis**

Before proceeding with a full scale investigation, it will be useful to do two things. First, an assessment for the number of required sample holders is needed. This will provide a range of values to consider during the design alternative search. The second issue is to examine the lower bounds on the required number of testers at each test cell. The desire is to prevent a woefully under-capacitated system from being examined (either in the Process Analyzer or in an OptQuest run). The execution of a woefully under-capacitated system may cause an execution error that may require the experiments to have to be redone.

Based on Table 11.10, in order to keep the testers busy in the high resource case, the system would need to have at least ( $2 + 2 + 3 + 4 + 5 = 16$ ) sample holders, one sample holder to keep each unit of each test cell busy. To investigate the effect of sample holders on the system, a set of experiments was run using the Process Analyzer on the high resource case with the number of sample holders steadily decreasing down to 16 sample holders. The results from this initial analysis are shown in Table 11.12. As can be seen in the table, the performance of the system degrades significantly if the number of sample holders goes below 16. In addition, it should be clear that for the high resource settings, the system is easily capable of meeting the design requirements. Also, it is clear that too many sample holders can be detrimental to the system time. Thus, a good range for the number of sample holders appears to be from 20 to 36.

The low resource case was also explored for the cases of 16 and 48 sample holders. The system appears to be highly under-capacitated at the low resource setting. Since the models were able to execute for the entire run length of 10 days, this provides evidence that any scenarios that have more resources will also be able to execute without any problems.

Based on the preliminary results, you should feel confident enough to design experiments and proceed with an analysis of the problem in order to make a recommendation. These experiments are described in the next section.

### **11.5 FINAL EXPERIMENTAL ANALYSIS AND RESULTS**

While the high capacity system has no problem achieving good system time performance, it will also be the most expensive configuration. Therefore, there is a clear trade-off between increased cost and improved customer service. Thus, a system configuration that has the minimum cost while also obtaining desired performance needs to be recommended. This situation can be analyzed in a number of ways using the Process Analyzer and OptQuest within the Arena™ environment. Often both tools are used to solve the problem. Typically, in an OptQuest analysis, an initial set of screening experiments may be run using the Process Analyzer to get a good idea on the bounds for the problem. Some of this analysis has already been done. See, for example, Table 11.12. Then, after an OptQuest analysis has

**TABLE 11.12 Initial Results**

Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	# Holders	Non-Rush Probability	Rush Probability	System Time
2	2	3	4	5	48	0.998	0.96	17.811
2	2	3	4	5	44	0.997	0.971	17.182
2	2	3	4	5	40	0.994	0.976	17.046
2	2	3	4	5	36	0.994	0.983	16.588
2	2	3	4	5	32	0.996	0.982	16.577
2	2	3	4	5	28	0.995	0.986	16.345
2	2	3	4	5	24	0.988	0.99	16.355
2	2	3	4	5	20	0.961	0.991	21.431
2	2	3	4	5	18	0.822	0.987	32.839
2	2	3	4	5	17	0.721	0.99	39.989
2	2	3	4	5	16	0.494	0.989	57.565
2	2	3	4	5	12	0	0.988	1309.683
1	1	2	2	3	16	0	0.985	2525.434
1	1	2	2	3	48	0	0.749	2234.676

been performed, additional experiments might be made via the Process Analyzer to hone the solution space. If only the Process Analyzer is to be used, then a carefully thought out set of experiments can do the job. For teaching purposes, this section will illustrate how to use both the Process Analyzer and OptQuest on this problem.

The contest problem also proposes the use of additional logic to improve the system's performance on rush samples. In a strict sense, this provides another factor (or design parameter) to consider during the analysis. The inclusion of another factor can dramatically increase the number of experiments to be examined. Because of this, an assumption will be made that the additional logic does not significantly interact with the other factors in the model. If this assumption is true, the original system can be optimized to find a basic design configuration. Then, the additional logic can be analyzed to check if it adds value to the basic solution. In order to recommend a solution, a trade-off between cost and system performance must be established. Since the problem specification desires that

From this simulation study, we would like to know what configuration would provide the most cost-effective solution while achieving high customer satisfaction. Ideally, we would always like to provide results in less time than the contract requires. However, we also do not feel that the system should include extra equipment just to handle the rare occurrence of a late report.

The objective is clear here: minimize total cost. In order to make the occurrence of a late report rare, limits can be set on the probability that the rush and non-rush sample's system times exceed the contract limits. This can be done by arbitrarily defining rare as 3 out of 100 samples being late. Thus, at least 97% of the non-rush and rush samples must meet the contract requirements.

### 11.5.1 Using the Process Analyzer on the Problem

This section uses the Process Analyzer on the problem within an experimental design paradigm. Further information on experimental design methods can be found in

**TABLE 11.13 First Experiment Factors and Levels**

Factor	Levels	
	Low	High
Cell 1 # units	1	2
Cell 2 # units	1	2
Cell 3 # units	2	3
Cell 4 # units	2	4
Cell 5 # units	3	5
# holders	20	36

Montgomery and Runger [2006]. For a discussion of experimental design within a simulation context, please see Law [2007] or Kleijnen [1998]. There are six factors (# units for each of the five testing cells and the number of sample holders). To initiate the analysis, a factorial experiment can be used. As shown in Table 11.13, the previously determined resource requirements can be readily used as the low and high settings for the test cells. In addition, the previously determined lower and upper range values for the number of sample holders can be used as the levels for the sample holders.

This amounts to  $2^6 = 64$  experiments; however, since sample holders have a cost, it makes sense to first run the half-fraction of the experiment associated with the lower level of the number of holders to see how the test cell resource levels interact. The results for the first half-fraction are shown in Table 11.14. The first readily apparent conclusion that can be drawn from this table is that test cells 1 and 2 must have at least two units of resource capacity. Now, considering cases (2, 2, 2, 4, 5) and (2, 2, 3, 4, 5), it is very likely that test cell # 3 requires 3 units of resource capacity in order to meet the contract requirements. Lastly, it is very clear that the low levels for cells 4 and 5 are not acceptable. Thus, the search range can be narrowed down to 3 and 4 units of capacity for cell 4 and to 4 and 5 units of capacity for cell 5.

The other half-fraction with the number of samples at 36 is presented in Table 11.15. The same basic conclusions concerning the resources at the test cells can be made by examining the results. In addition, it is apparent that the number of sample holder can have a significant effect on the performance of the system. It appears that more sample holders hurt the performance of the under-capacitated configurations. The effect of the number of sample holders for the highly capacitated systems is somewhat mixed, but generally, the results indicate that 36 sample holders are probably too many for this system. Of course, since this has all been done within an experimental design context, more rigorous statistical tests can be performed to fully test these conclusions. These tests will not do that here, but rather the results will be used to set up another experimental design that can help to better examine the system's response.

Using the initial results in Table 11.12 and the analysis of the two half-fraction experiments, another set of experiments were designed to focus in on the capacities for cells 3, 4, and 5. The experiments are given in Table 11.16. This set of experiments is a  $2^4 = 16$  factorial experiment. The results are shown in Table 11.17. The results have been sorted such that the systems that have the higher chance of meeting the contract requirements are at the top of the table. From the results, it should be clear that cell 3 requires at least 3 units of capacity for the system to be able to meet the requirements. It is also very likely that cell

**TABLE 11.14 Half-Fraction with # Sample Holders = 20**

Cell Resource Units					# Holders	Probability		System Time	Total Cost
1	2	3	4	5		Non-Rush	Rush		
1	1	2	2	3	20	0	0.963	2410.517	100,340
2	1	2	2	3	20	0	0.966	2333.135	110,340
1	2	2	2	3	20	0	0.98	1913.694	112,740
2	2	2	2	3	20	0	0.973	1904.499	122,740
1	1	3	2	3	20	0	0.966	2373.574	108,840
2	1	3	2	3	20	0	0.962	2407.404	118,840
1	2	3	2	3	20	0	0.973	1902.445	121,240
2	2	3	2	3	20	0	0.978	1865.821	131,240
1	1	2	4	3	20	0	0.951	2332.412	119,940
2	1	2	4	3	20	0	0.949	2365.047	129,940
1	2	2	4	3	20	0.003	0.985	496.292	132,340
2	2	2	4	3	20	0.025	0.986	284.205	142,340
1	1	3	4	3	20	0	0.956	2347.05	128,440
2	1	3	4	3	20	0	0.956	2316.931	138,440
1	2	3	4	3	20	0.019	0.984	364.081	140,840
2	2	3	4	3	20	0.122	0.987	157.798	150,840
1	1	2	2	5	20	0	0.968	2394.53	122,740
2	1	2	2	5	20	0	0.965	2360.751	132,740
1	2	2	2	5	20	0	0.98	1873.405	135,140
2	2	2	2	5	20	0	0.982	1865.02	145,140
1	1	3	2	5	20	0	0.963	2391.649	131,240
2	1	3	2	5	20	0	0.965	2361.708	141,240
1	2	3	2	5	20	0	0.974	1926.889	143,640
2	2	3	2	5	20	0	0.98	1841.387	153,640
1	1	2	4	5	20	0	0.951	2387.81	142,340
2	1	2	4	5	20	0	0.955	2352.933	152,340
1	2	2	4	5	20	0.202	0.986	121.199	154,740
2	2	2	4	5	20	0.683	0.991	43.297	164,740
1	1	3	4	5	20	0	0.954	2291.88	150,840
2	1	3	4	5	20	0	0.947	2332.031	160,840
1	2	3	4	5	20	0.439	0.985	69.218	163,240
2	2	3	4	5	20	0.961	0.991	21.431	173,240

4 requires 4 testers to meet the requirements. Thus, the search space has been narrowed to either 4 or 5 testers at cell 5 and between 20 and 24 holders.

To finalize the selection of the best configuration for the current situation, 10 experimental combinations of cell 5 resource capacity (4, 5) and number of sample holders (20, 21, 22, 23, 24) were run in the Process Analyzer using 10 replications for each scenario. The results of the experiments are shown in Table 11.18. In addition, the multiple comparison procedure was applied to the results based on picking the solution with the best (highest) non-rush probability. The results of that comparison are shown in Figure 11.20. The multiple comparison procedure indicates with 95% confidence that scenarios 3, 4, 5, 7, 8, 9, 10

**TABLE 11.15 Half-Fraction with # Sample Holders = 36**

Cell Resource Units					# Holders	Probability		System Time	Total Cost
1	2	3	4	5		Non-Rush	Rush		
1	1	2	2	3	36	0	0.776	2319.323	106,532
2	1	2	2	3	36	0	0.764	2260.829	116,532
1	2	2	2	3	36	0	0.729	1816.568	118,932
2	2	2	2	3	36	0	0.714	1779.53	128,932
1	1	3	2	3	36	0	0.768	2238.654	115,032
2	1	3	2	3	36	0	0.775	2243.56	125,032
1	2	3	2	3	36	0	0.73	1762.579	127,432
2	2	3	2	3	36	0	0.717	1763.306	137,432
1	1	2	4	3	36	0	0.796	2244.243	126,132
2	1	2	4	3	36	0	0.79	2268.549	136,132
1	2	2	4	3	36	0.202	0.9	111.232	138,532
2	2	2	4	3	36	0.385	0.915	72.86	148,532
1	1	3	4	3	36	0	0.806	2255.19	134,632
2	1	3	4	3	36	0	0.795	2251.548	144,632
1	2	3	4	3	36	0.36	0.899	76.398	147,032
2	2	3	4	3	36	0.405	0.911	68.676	157,032
1	1	2	2	5	36	0	0.771	2304.784	128,932
2	1	2	2	5	36	0	0.771	2250.198	138,932
1	2	2	2	5	36	0	0.738	1753.819	141,332
2	2	2	2	5	36	0	0.717	1751.676	151,332
1	1	3	2	5	36	0	0.78	2251.174	137,432
2	1	3	2	5	36	0	0.771	2252.106	147,432
1	2	3	2	5	36	0	0.726	1781.024	149,832
2	2	3	2	5	36	0	0.716	1794.93	159,832
1	1	2	4	5	36	0	0.787	2243.338	148,532
2	1	2	4	5	36	0	0.794	2243.558	158,532
1	2	2	4	5	36	0.541	0.897	54.742	160,932
2	2	2	4	5	36	0.898	0.932	28.974	170,932
1	1	3	4	5	36	0	0.791	2263.484	157,032
2	1	3	4	5	36	0	0.799	2278.707	167,032
1	2	3	4	5	36	0.604	0.884	49.457	169,432
2	2	3	4	5	36	0.994	0.983	16.588	179,432

**TABLE 11.16 Second Experiment Factors and Levels**

Factor	Levels	
	Low	High
Cell 3 # units	2	3
Cell 4 # units	3	4
Cell 5 # units	4	5
# holders	20	24

**TABLE 11.17 Half-Fraction with # Sample Holders = 20**

Cell Resource Units					# Holders	Probability		System Time	Total Cost
1	2	3	4	5		Non-Rush	Rush		
2	2	3	4	5	24	0.988	0.99	16.355	174,788
2	2	3	4	4	24	0.97	0.988	19.41	163,588
2	2	3	4	5	20	0.961	0.991	21.431	173,240
2	2	3	4	4	20	0.945	0.989	25.27	162,040
2	2	3	3	4	24	0.877	0.987	30.345	153,788
2	2	3	3	5	24	0.871	0.988	31.244	164,988
2	2	2	4	5	24	0.812	0.984	34.065	166,288
2	2	2	4	4	24	0.782	0.986	35.289	155,088
2	2	3	3	5	20	0.734	0.991	39.328	163,440
2	2	3	3	4	20	0.731	0.992	40.37	152,240
2	2	2	3	4	24	0.687	0.987	42.891	145,288
2	2	2	4	5	20	0.683	0.991	43.297	164,740
2	2	2	3	5	24	0.656	0.98	46.049	156,488
2	2	2	4	4	20	0.627	0.989	45.708	153,540
2	2	2	3	5	20	0.561	0.987	52.71	154,940
2	2	2	3	4	20	0.492	0.985	59.671	143,740

**TABLE 11.18 Results for the Final Set of Experiments**

Scenario	Cell Resource Units					# Holders	Probability		System Time	Total Cost
	1	2	3	4	5		Non-Rush	Rush		
1	2	2	3	4	4	20	0.917	0.987	26.539	162,040
2	2	2	3	4	4	21	0.942	0.987	23.273	162,427
3	2	2	3	4	4	22	0.976	0.989	20.125	162,814
4	2	2	3	4	4	23	0.974	0.988	19.983	163,201
5	2	2	3	4	4	24	0.979	0.989	18.512	163,588
6	2	2	3	4	5	20	0.958	0.988	21.792	173,240
7	2	2	3	4	5	21	0.967	0.987	19.362	173,627
8	2	2	3	4	5	22	0.984	0.988	18.022	174,014
9	2	2	3	4	5	23	0.986	0.988	16.971	174,401
10	2	2	3	4	5	24	0.98	0.987	16.996	174,788

are the best. Since there is essentially no difference between them, the cheapest configuration can be chosen, which is scenario 3 with factor levels of (2, 3, 3, 4, 4, and 22) and a total cost of \$162,814.

Based on the results of this section, a solid solution for the problem has been determined; however, to give you experience in applying OptQuest to a problem, let us examine its application to this situation.

### 11.5.2 Using OptQuest on the Problem

As shown in Figure 11.21, OptQuest for Arena™ can be used to define controls and responses and to setup an optimization model to minimize total cost subject to the

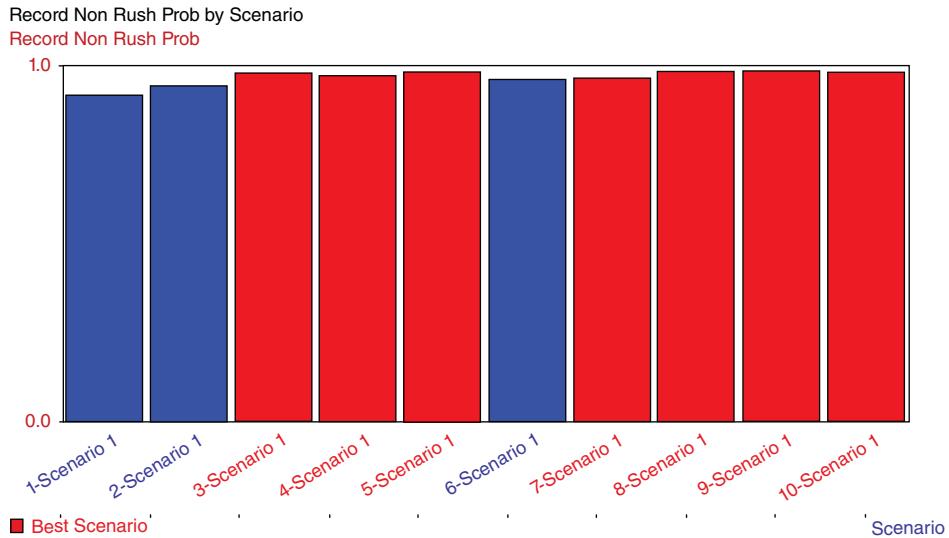


Figure 11.20 Multiple comparison results for non-rush probability.

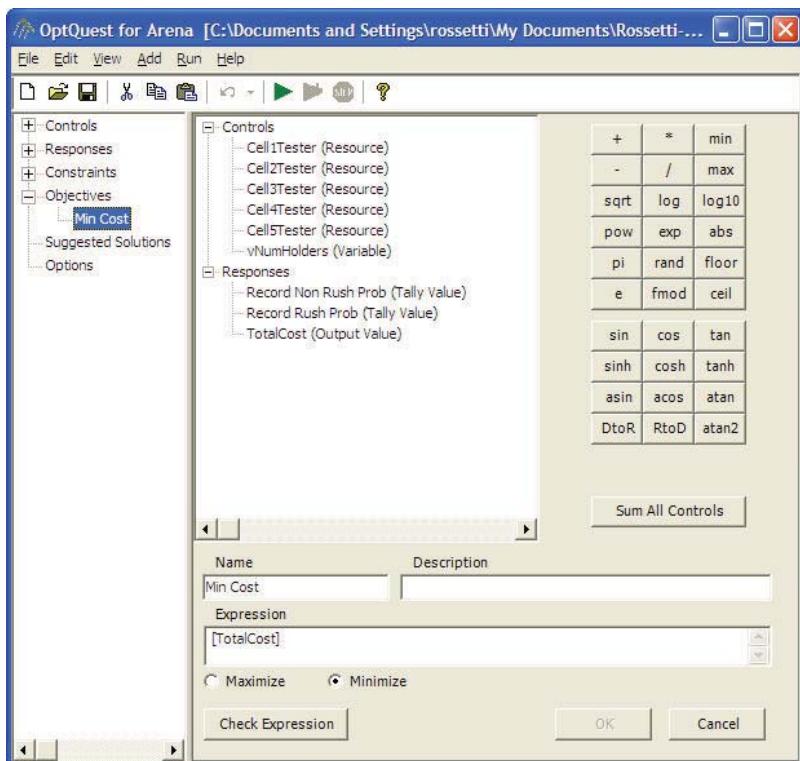


Figure 11.21 OptQuest objective definition.

probability of non-rush and rush samples meeting the contract requirements being greater than or equal to 0.97. When running OptQuest, it is a good idea to try to give OptQuest a good starting point. Since the high resource case is already known to be feasible from the pilot experiments, the optimization can be started using the high resource case and the number of sample holders within the middle of their range (e.g., 27 sample holders).

The simulation optimization was executed over a period of about 6 hours (wall clock time) until it was manually stopped after 79 total simulations. From Figure 11.22, the best solution was 2 testers at cells 1 and 2, 3 testers at cell 3, 4 testers at cells 4 and 5, and 23 sample holders. The total cost of this configuration is \$163,201. This solution has 1 more sample holder as compared to the solution based on the Process Analyzer. Of course, you would not know this if the analysis was based solely on OptQuest. OptQuest allows top solutions to be saved to a file. Upon looking at those solutions, it becomes clear that the heuristic in OptQuest has found the basic configuration for the test cells (2, 2, 3, 4, 4) and is investigating the number of sample holders. If the optimization had been allowed to run longer, it is very likely that it would have recommended 22 sample holders. At this stage, the Process Analyzer can be used to hone in on the recommended OptQuest solution by more closely examining the final set of “best” solutions. Since that has already been done in the previous section, it will not be included here.

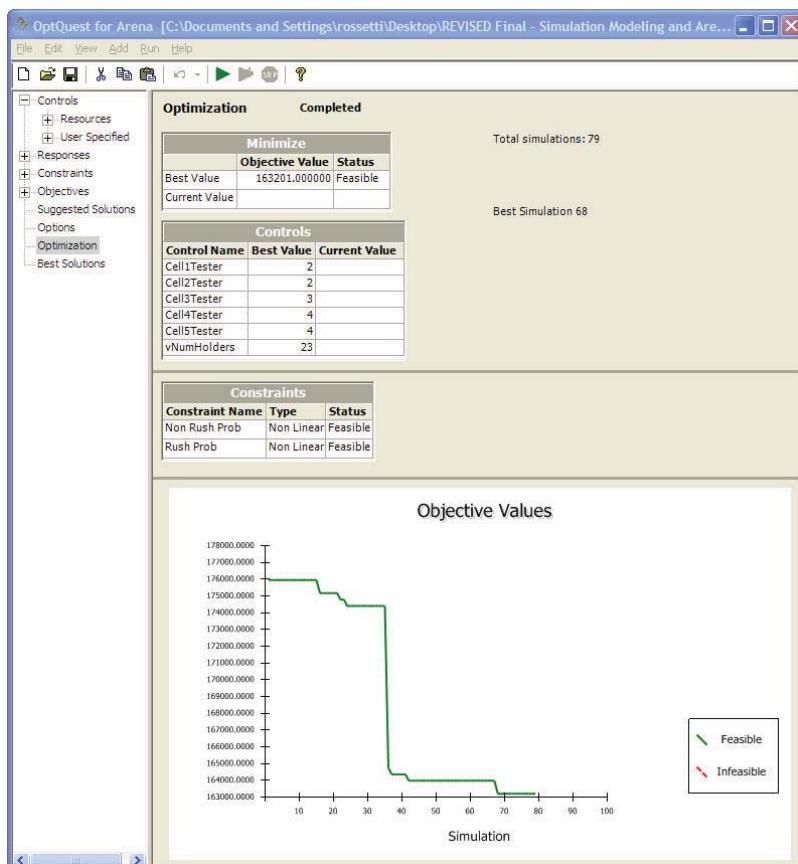


Figure 11.22 OptQuest result.

### 11.5.3 Investigating the New Logic Alternative

Now that a recommended basic configuration is available, the alternative logic needs to be checked to see if it can improve performance for the additional cost. In addition, the suggested number for the control logic should be determined. The basic model can be easily modified to contain the alternative logic. This was done within the load/unload submodel. In addition, a flag variable was used to be able to turn on or turn off the use of the new logic so that the Process Analyzer can control whether or not the logic is included in the model. The implementation can be found in the file *SMTTesting.doe*.

To reduce the number of factors involved in the previous analysis, it was assumed that the new logic did not interact significantly with the allocation of the test cell capacity; however, because the suggested number checks for how many samples are waiting at the load/unload station, there may be some interaction between these factors. Based on these assumptions, 10 scenarios with the new logic were designed for the recommended tester configuration (2, 2, 3, 4, 4) varying the number of holders between 20 and 24 with the suggested number (SN) set at both 2 and 3. Table 11.19 presents the results of these experiments. From the results, it is clear that the new logic does not have a substantial impact on the probability of meeting the contract for the non-rush and rush samples. In fact, looking at scenario 3, the new logic may actually hurt the non-rush samples. To complete the analysis, a more rigorous statistical comparison should be performed; however, that task will be skipped for the sake of brevity.

After all the analysis, the results indicate that SMTTesting should proceed with the (2, 2, 3, 4, 4) configuration with 22 sample holders for a total cost of \$162, 814. The additional one-time purchase of the control logic is not warranted at this time.

## 11.6 SENSITIVITY ANALYSIS

This realistically sized case study clearly demonstrates the application of simulation to design a manufacturing system. The application of simulation was crucial in this analysis because of the complex, dynamic relationships between the elements of the system and because of the inherent stochastic environment (e.g., arrivals and breakdowns) that were in the problem. Based on the analysis performed for the case study, SMTTesting should

**TABLE 11.19 Results for Analyzing the New Logic**

Scenario	Cell Resource Units					# Holders	SN	Probability		System Time
	1	2	3	4	5			Non-Rush	Rush	
1	2	2	3	4	4	20	2	0.937	0.989	23.937
2	2	2	3	4	4	21	2	0.956	0.988	21.461
3	2	2	3	4	4	22	2	0.957	0.989	21.159
4	2	2	3	4	4	23	2	0.972	0.989	19.011
5	2	2	3	4	4	24	2	0.975	0.986	18.665
6	2	2	3	4	4	20	3	0.926	0.989	25.305
7	2	2	3	4	4	21	3	0.948	0.988	22.449
8	2	2	3	4	4	22	3	0.968	0.989	20.536
9	2	2	3	4	4	23	3	0.983	0.988	18.591
10	2	2	3	4	4	24	3	0.986	0.987	18.17

be confident that the recommended design will suffice for the current situation. However, there are a number of additional steps that can (and probably should) be performed for the problem.

In particular, to have even more confidence in the design, simulation offers the ability to perform a sensitivity analysis on the “uncontrollable” factors in the environment. While the analysis in the previous section concentrated on the design parameters that can be controlled, it is very useful to examine the sensitivity of other factors and assumptions on the recommended solution. Now that a verified and validated model is in place, the arrival rates, the breakdown rates, repair times, the buffer sizes, etc. can all be varied to see how they affect the recommendation. In addition, a number of explicit and implicit assumptions were made during the modeling and analysis of the contest problem. For example, when modeling the breakdowns of the testers, the FAILURE module was used. This module assumes that when a failure occurs, all units of the resource become failed. This may not actually be the case. In order to model individual unit failures, the model must be modified. After the modification, the assumption can be tested to see if it makes a difference with respect to the recommended solution. Other aspects of the situation were also omitted. For example, the fact that tests may have to be repeated at test cells was left out due to the fact that no information was given as to the likelihood for repeating the test. This situation could be modeled and a retest probability assumed. Then, the sensitivity to this assumption could be examined.

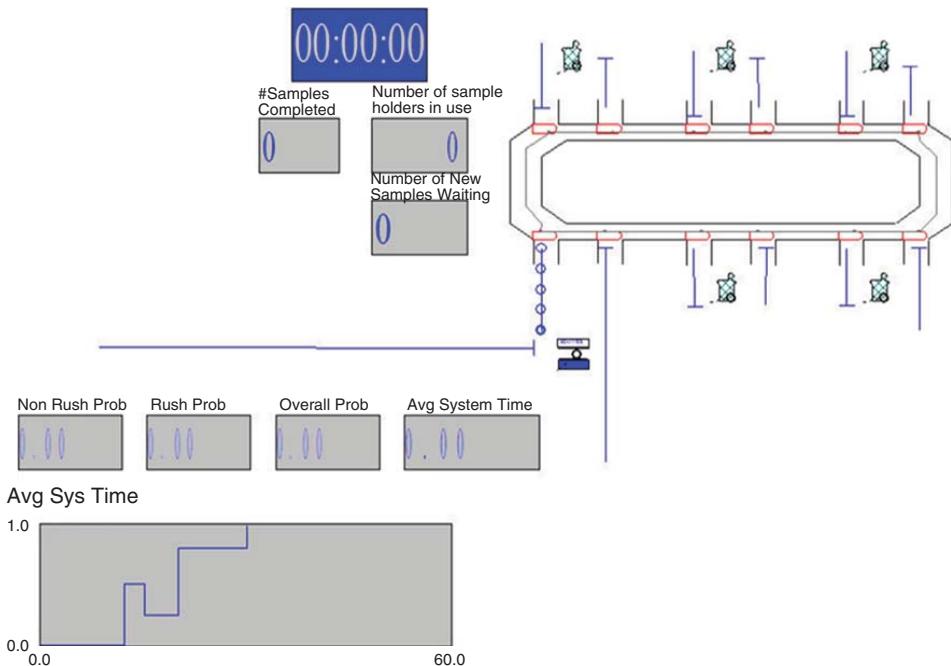
In addition to modeling assumptions, assumptions were made during the experimental analysis. For example, basic resource configuration of (2, 2, 3, 4, 4) was assumed to not change if the control logic was introduced. This can also be tested within a sensitivity analysis. Since performing a sensitivity analysis has been discussed in Chapter 7, the mechanics of performing sensitivity analysis will not be discussed here. A number of exercises will ask you to explore these issues.

## 11.7 COMPLETING THE PROJECT

A major aspect of the use of simulation is convincing the key decision makers concerning the recommendations of the study. Developing an animation for the Arena™ model is especially useful in this regard. For the SMTesting model, an animation is available within the file. The animation (Figure 11.23) animates the conveyor, the usage of the resources, queues, and has a number of variables for monitoring the system. In addition, user-defined views were defined for easily viewing the animation and the model logic.

A written report is also a key requirement of a simulation project. Chung [2004] provides some guidelines for the report and a presentation associated with a project. An outline for a simulation project report is given below.

1. Executive Summary
  - (a) Provide a brief summary of the problem, method, results, and recommendation. It should be oriented toward the decision maker.
  - (b) A good executive summary should have the following characteristics: Essential facts of problem provided; methods of analysis summarized succinctly; recommendations clearly tied to results, clear decision recommended; essential trade-offs provided in terms of key performance measures; decision maker knows what to do.



**Figure 11.23** Overview of the animation.

## 2. Introduction

- (a) Provide a brief overview of the system and problem context. Orient the reader as to what should be expected from the report.

## 3. System and Problem Description

- (a) Describe the system under study. The system should be described in such a way that an analyst could develop a model from the description. Describe the problem to be solved. Indicate the key performance metrics to be used to make the decision and the major goals/objectives to be achieved. For example, the contest problem description given in this chapter is an excellent example of describing the system and the problem.

## 4. Solution and Modeling Approach

- (a) Describe the modeling issues that needed to be solved. Describe the inputs for the model and how they were developed. For each major modeling issue, you should describe briefly your conceptual approach. Relate your description to the system description. Provide activity diagrams or pseudo-code and describe how this relates to your conceptual model. Describe the major outputs from the model and how they are collected.
- (b) Describe the overall flow of the model and relate this to your conceptual model. Give an accounting of important modeling logic where the detail is necessary to help the reader understand a tricky issue or important modeling issue.
- (c) State relevant and important assumptions. Discuss their potential effect on your answers and any procedures you took to test your assumptions.

- (d) Describe any verification/validation procedures. Describe your up-front analysis. You might approximate parts of your model or do some basic calculations that give you a ballpark idea of the range of your outputs. If you have data from the system concerning expected outputs, then you should compare the simulation output to the system output using statistical techniques. For a more detailed discussion of these techniques, please see the excellent discussion in Balci [1998a].

## 5. Experimentation and Analysis

- (a) Describe a typical simulation run. Is the model a finite horizon or steady-state model? Analyzed by replication method or batch means? If it is a steady-state simulation, how did you handle the initial transient period? How did you determine the run length? The number of replications? Describe or reference any special statistical methods that you used so that the reader can understand your results.
- (b) Describe your experimental plan and the results. What are your factors and the levels? What are you varying in the model? What are you capturing as output? Describe the results from your experiments. Use figures and tables and refer to them in your text to explain the meaning of the results. Discuss the effect of assumptions and sensitivity analysis and their effect on the model's results.

## 6. Conclusions and Recommendations

- (a) Give the answers to the problem. Back up your recommendations based on your results and analysis. If possible, give the solution in the form of alternatives. For example, "Assuming X, we recommend Y because of Z" or "Given results X, Y, we recommend Z"

## 7. References

## 8. Appendix

- (a) Supporting materials, only if absolutely necessary.

In addition to the above material, a simulation report might also contain plans for how to implement the recommendations. A presentation covering the major issues and recommendations from the report should also be prepared. You can think of the presentation as a somewhat expanded version of the executive summary of the report. It should concentrate on building credibility for your analysis, the results, and the recommendations. You should consider using your animation during the presentation as a tool for building credibility and understanding for what was modeled.

Besides the model, written report, presentation, and animation, it is extremely useful to provide model/user documentation. To assist with some of the documentation, you might investigate the Model Documentation Report available on the Tools menu. This tool produces an organized listing of the modules within the model. However, this tool is no substitute for fully describing the model in a language-independent manner within your report. Even if nonsimulation-oriented users will not be interacting with the model, it is very useful to have a user's guide to the model of some sort. This document should describe how to run the model, what inputs to modify, and what outputs to examine, etc. Often a model built for a single use is rediscovered and used for additional analysis. A user's guide becomes extremely useful in these situations.

## 11.8 SOME FINAL THOUGHTS

This chapter presented how to apply simulation to a realistic problem. The solution involved many of the simulation modeling techniques that you learned throughout the text. In addition, Arena<sup>TM</sup> and its supporting tools were applied to develop a model for the problem and to analyze the system under study. Finally, experimental design concepts were applied in order to recommend design alternatives for the problem. This comprehensive example should give you a good understanding for how to apply simulation in the *real* world.

There are a number of other issues in simulation that have not been given full description within this text. However, based on the material in this text, you should be well prepared to apply simulation to problems in engineering and business. To continue your education in simulation, you should consider reading Chung [2004] and Banks [1998]. Chapter 13 of Chung [2004] provides a number of examples of the application simulation, and Chapters 14–21 of Banks [1998] illustrates simulation in areas such as health care, the military, logistics, and manufacturing. In addition, Chapters 22 and 23 of Banks [1998] provide excellent practical advice on how to perform and manage a successful simulation project.

Finally, you should get involved in professional societies and organizations that support simulation modeling and analysis, such as

- American Statistical Association (ASA)
- Association for Computing Machinery: Special Interest Group on Simulation (ACM/SIGSIM)
- Institute of Electrical and Electronics Engineers: Systems, Man, and Cybernetics Society (IEEE/SMC)
- Institute for Operations Research and the Management Sciences: Simulation Society (INFORMS-SIM)
- Institute of Industrial Engineers (IIE)
- National Institute of Standards and Technology (NIST)
- The Society for Modeling and Simulation International (SCS).

An excellent opportunity to get involved is through the Winter Simulation Conference. You can find information concerning these organizations and the Winter Simulation Conference easily through the Internet.

As a final thought you should try to adhere to the following rules during your practice of simulation.

1. To only perform simulation studies that have clearly defined objectives and performance metrics.
2. To never perform a simulation study when the problem can be solved through simpler and more direct techniques.
3. To always carefully plan, collect, and analyze the data necessary to develop good input distributions for simulation models.
4. To never accept simulation results or decisions based on a sample size of 1.
5. To always use statistically valid methods to determine the sample size for simulation experiments.
6. To always check that statistical assumptions (e.g., independence and normality) are reasonably valid before reporting simulation results.

7. To always provide simulation results by reporting the sample size, the sample average, and a measure of sample variation (e.g., sample variance, sample standard deviation, standard error, and sample half-width for a specified confidence level)
8. To always perform simulation experiments with a random number generator that has been tested and proven to be reliable.
9. To always take steps to mitigate the effects of initialization bias when performing simulations involving steady-state performance.
10. To always verify simulation models through techniques such as debugging, event tracing, and animation.
11. To always validate simulation models through common sense analytical thinking techniques, discussions with domain experts, real data, and sensitivity analysis techniques.
12. To always carefully plan, perform, and analyze simulation experiments using appropriate experimental design techniques.

By following these rules, you should have a successful and professional simulation career.

## EXERCISES

- 11.1 Perform a sensitivity analysis on the recommendation for the SMTTesting contest problem by varying the arrival rate for each hour by  $\pm 10\%$ . In addition, examine the effect of decreasing or increasing the time between failures for test cells 1, 3, 4, 5 by  $\pm 10\%$  as well as the mean time for repair.
- 11.2 What solution should be recommended if the probability of meeting the contract requirements is specified as 95% for non-rush and rush samples?
- 11.3 Revise the model to handle the independent failure of individual testers at each test cell. Does the more detailed modeling change the recommended solution?
- 11.4 Revise the model to handle the situation of retesting at each test cell. Assume that a retest occurs with a 3% chance and then test the sensitivity of this assumption.
- 11.5 How does changing the buffer capacities for the load/unload area and the test cells affect the recommended solution?
- 11.6 Develop a model that separates the load/unload area into two stations, one for loading and one for unloading with each station having its own machine. Analyze where to place the stations on the conveyor and compare your design to the recommended solution for the contest problem.
- 11.7 Revise the model so that it uses the BATCH/SEPARATE approach to sample holder and sample interaction rather than the PICKUP/DROPOFF approach. Compare the results of your model to the results presented in this chapter.
- 11.8 Solve one of the Rockwell Software/IIE contest problems.

## BIBLIOGRAPHY

- J. Ahrens and V. Dieter. Computer methods for sampling from the exponential and normal distributions. *Communications of the Association for Computing Machinery*, 15:873–882, 1972.
- Air Force Systems Command. ASD Directorate of Systems Engineering and DSMC Technical Management Department. 1991.
- S. C. Albright. *VBA for Modelers, Developing Decision Support Systems with Microsoft Excel*. Duxbury Thomson Learning, 2001.
- C. Alexopoulos and A. F. Seila. Output data analysis. In J. Banks, editor, *Handbook of Simulation*. John Wiley & Sons, New York, 1998.
- J. April, F. Glover, J. Kelly, and M. Laguna. Simulation optimization using real-world applications. In B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, editors, *The Proceedings of the 2001 Winter Simulation Conference*. Institute of Electrical and Electronic Engineers, Piscataway, NJ, 2001.
- R. G. Askin and J. B. Goldberg. *Design and Analysis of Lean Production Systems*. John Wiley & Sons, 2002.
- R. G. Askin and C. R. Standridge. *Modeling and Analysis of Manufacturing Systems*. John Wiley & Sons, 1993.
- S. Axssäter. *Inventory Control*. Springer Science + Business Media, 2006.
- O. Balci. Principles of simulation model validation, verification, and testing. *Transactions of the Society for Computer Simulation International*, 14(1):3–12, 1997.
- O. Balci. Verification, validation, and testing. In *The Handbook of Simulation*, pages 335–393. John Wiley & Sons, 1998a.
- O. Balci. Verification, validation, and accreditation. In D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, editors, *Proceedings of the 1998 Winter Simulation Conference*, pages 41–48. IEEE Piscataway, New York, 1998b.
- R. H. Ballou. *Business Logistics/Supply Chain Management: Planning, Organizing, and Controlling the Supply Chain*. Prentice-Hall, 5th edition, 2004.

- J. Banks. *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. John Wiley & Sons, 1998.
- J. Banks, B. Burnette, H. Kozloski, and J. Rose. *Introduction to SIMAN V and CINEMA V*. John Wiley & Sons, 1995.
- J. Banks, J. Carson, B. Nelson, and D. Nicol. *Discrete-Event System Simulation*. Prentice-Hall, 4th edition, 2005.
- B. Biller and B. L. Nelson. Modeling and generating multivariate time-series input processes using a vector autoregressive technique. *Association for Computing Machinery Transactions on Modeling and Computer Simulation*, 13:211–237, 2003.
- B. S. Blanchard and W. J. Fabrycky. *Systems Engineering and Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1990.
- G. Bolch, S. Greiner, H. de Meer, and K. Trivedi. *Queueing Networks and Markov Chains*. John Wiley & Sons, 2nd edition, 2006.
- G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice-Hall, 3rd edition, 1994.
- G. E. P. Box and M. F. Muller. Note on the generation of random normal deviates. *Annals of Mathematical Statistics*, 29:610–11, 1958.
- J. A. Buzacott and J. G. Shanthikumar. *Stochastic Models of Manufacturing Systems*. Prentice-Hall, 1993.
- M. C. Cario and B. L. Nelson. Autoregressive to anything: time series input processes for simulation. *Operations Research Letters*, 19:51–58, 1996.
- M. C. Cario and B. L. Nelson. Numerical methods for fitting and simulating autoregressive-to-anything processes. *INFORMS Journal of Computing*, 10:72–81, 1998.
- G. Casella and R. Berger. *Statistical Inference*. Wadsworth & Brooks/Cole, 1990.
- C. R. Cash, D. G. Dippold, J. M. Long, B. L. Nelson, and W. P. Pollard. Evaluation of tests for initial conditions bias. In J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, editors, *Proceedings of the 1992 Winter Simulation Conference*, pages 577–585. Institute of Electrical and Electronics Engineers, Piscataway, NJ, 1992.
- R. C. Cheng. The generation of gamma variables with nonintegral shape parameters. *Applied Statistics*, 26(1):71–75, 1977.
- S. Chopra and P. Meindl. *Supply Chain Management: Strategy, Planning, and Operations*. Prentice-Hall, 3rd edition, 2007.
- C. A. Chung. *Simulation Modeling Handbook: A Practical Approach*. CRC Press, 2004.
- R. B. Cooper. *Introduction to Queueing Theory*. CEEPress Books, 3rd edition, 1990.
- L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, 1986.
- G. S. Fishman. *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer, New York, 2001.
- G. S. Fishman. *A First Course in Monte Carlo*. Thomson Brooks/Cole, 2006.
- G. S. Fishman and L. S. Yarberry. An implementation of the batch means method. *INFORMS Journal on Computing*, 9:296–310, 1997.
- F. Glover, J. P. Kelly, and M. Laguna. New advances for welding optimization and simulation. In F. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, editors, *The Proceedings of the 1999 Winter Simulation Conference*. Institute of Electrical and Electronic Engineers, Piscataway, NJ, 1999.
- F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, 1997.
- P. W. Glynn and W. Whitt. Extensions of the queueing relation and  $L = \lambda W$  and  $H = \lambda G$ . *Operations Research*, 37:634–644, 1989.
- D. Goldsman and B. L. Nelson. Comparing systems via simulation. In J. Banks, editor, *Handbook of Simulation*. John Wiley & Sons, New York, 1998.

- D. Gross and C. M. Harris. *Fundamentals of Queueing Theory*. John Wiley & Sons, New York, 3rd edition, 1998.
- D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris. *Fundamentals of Queueing Theory*. John Wiley & Sons, 2008.
- G. Hadley and T. M. Whitin. *Analysis of Inventory Systems*. Prentice-Hall, 1963.
- J. Henriksen and T. Schriber. Simplified approaches to modeling accumulating and non-accumulating conveyor systems. In *Proceedings of the 1986 Winter Simulation Conference*. Institute of Electrical and Electronic Engineers, 1986.
- T. E. Hull and A. R. Dobell. Random number generators. *SIAM Review*, 4:230–254, 1962.
- F. P. Kelly. *Reversibility and Stochastic Networks*. John Wiley & Sons, 1979.
- W. D. Kelton, R. P. Sadowski, and D. T. Sturrock. *Simulation with Arena*. McGraw-Hill, 3rd edition, 2004.
- D. G. Kendall. Stochastic processes occurring in the theory of queues and their analysis by the method of imbedded markov chains. *Annals of Mathematical Statistics*, 24:338–354, 1953.
- J. F. C. Kingman. The heavy traffic approximation in the theory of queues. In *Proceedings of the Symposium on Congestion Theory*, 1964.
- J. P. C. Kleijnen. Analyzing simulation experiments with common random numbers. *Management Science*, 34:65–74, 1988.
- J. P. C. Kleijnen. Experimental design for sensitivity analysis, optimization, and validation of simulation models. In J. Banks, editor, *Handbook of Simulation*. John Wiley & Sons, New York, 1998.
- L. Kleinrock. *Queueing Systems*, volume 1. John Wiley & Sons, 1975.
- E. K. Lada, J. R. Wilson, and N. M. Steiger. A wavelet-based spectral method for steady-state simulation analysis. In *Proceedings of the 2003 Winter Simulation Conference*, 2003.
- A. Law. *Simulation Modeling and Analysis*. McGraw-Hill, 4th edition, 2007.
- L. J. LeBlanc and M. R. Galbreth. Verifying documentation standards in spreadsheet analysis. *Decision Sciences Journal of Innovative Education*, 7(1):81–88, 2009.
- P. L'Ecuyer. Testing random number generators. In *The Proceedings of the 1992 Winter Simulation Conference*, pages 305–313. Institute of Electrical and Electronic Engineers, 1992.
- P. L'Ecuyer, R. Simard, and W. D. Kelton. An object-oriented random number package with many long streams and substreams. *Operations Research*, 50:1073–1075, 2002.
- L. Leemis. Nonparametric estimation of the cumulative intensity function for a nonhomogeneous poisson process. *Management Science*, 37(7):886–900, 1991.
- L. M. Leemis and S. K. Park. *Discrete-Event Simulation: A First Course*. Prentice-Hall, 2006.
- J. D. C. Little. A proof for the queuing formula  $L = \lambda W$ . *Operations Research*, 9:383–387, 1961.
- J. R. Litton and C. H. Harmonosky. A comparison of selective initialization bias elimination methods. In *Proceeding of the 2002 Winter Simulation Conference*, 2002.
- M. Livny, B. Melamed, and A. K. Tsiolis. The impact of autocorrelation on queueing systems. *Management Science*, 39(3):322–339, 1993.
- B. D. McCullough and D. A. Heiser. On the accuracy of statistical procedures in microsoft excel. *Computational Statistics and Data Analysis*, 52:4570–4578, 2008.
- L. H. Miller. Table of percentage points of kolmogorov statistics. *Journal of the American Statistical Association*, 51(273):111–121, 1956.
- D. C. Montgomery and G. C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley & Sons, 4th edition, 2006.
- J. A. Muckstadt and A. Sapras. *Principles of Inventory Management: When You Are Down to Four, Order More*. Springer, 2010.
- S. Nahmias. *Production and Operations Analysis*. McGraw-Hill, 4th edition, 2001.

- B. E. Patuwo, R. L. Disney, and D. C. Mcnickle. The effect of correlated arrivals on queues. *IIE Transactions*, 25(3):105–110, 1993.
- C. D. Pegden, R. E. Shannon, and R. P. Sadowski. *Introduction to Simulation Using SIMAN*. McGraw-Hill, 2nd edition, 1995.
- S. G. Powell, K. R. Baker, and B. Lawson. A critical review of the literature on spreadsheet errors. *Decision Support Systems*, 46(1):128–138, 2008.
- K. Rajalingham, D. Chadwick, B. Knight, and D. Edwards. Quality control in spreadsheets: a software engineering-based approach to spreadsheet development. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, 2000.
- A. Ravindran, D. Phillips, and J. Solberg. *Operations Research Principles and Practice*. John Wiley & Sons, 2nd edition, 1987.
- B. D. Ripley. *Stochastic Simulation*. John Wiley & Sons, 1987.
- S. Robinson. Automated analysis of simulation output data. *Proceedings of the 2005 Winter Simulation Conference*, 763–770, 2005.
- S. Ross. *Introduction to Probability Models*. Academic Press, 6th edition, 1997.
- M. D. Rossetti. JSL: an open-source object-oriented framework for discrete-event simulation in Java. *International Journal of Simulation and Process Modeling*, 4(1):69–87, 2008.
- M. D. Rossetti and P. J. Delaney. *Control of Initialization Bias in Queueing Simulations Using Queueing Approximations*. Institute of Electrical and Electronics Engineers, Piscataway, NJ, 1995.
- M. D. Rossetti and Z. Li. Exploring exponentially weighted moving average control charts to determine the warm-up period. In M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, editors, *Proceedings of the 2005 Winter Simulation Conference*, pages 771–780. Institute of Electrical and Electronics Engineers, Piscataway, NJ, 2005.
- M. D. Rossetti and F. Seldanari. Multi-objective analysis of Hospital Delivery Systems. *Computers and Industrial Engineering*, 41:309–333, 2001.
- B. W. Schmeiser. Batch size effects in the analysis of simulation output. *Operations Research*, 30:556–568, 1982.
- T. J. Schriber and D. T. Brunner. How discrete-event simulation software works. In J. Banks, editor, *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, chapter 24. John Wiley & Sons, New York, 1998.
- L. Schruben. Detecting initialization bias in simulation output. *Operations Research*, 30:569–590, 1982.
- L. Schruben, H. Singh, and L. Tierney. Optimal tests for initialization bias in simulation output. *Operations Research*, 31:1167–1178, 1983.
- E. A. Silver, D. F. Pyke, and R. Peterson. *Inventory Management and Production Planning and Scheduling*. John Wiley & Sons, 3rd edition, 1998.
- N. Singh. *Systems Approach to Computer-Integrated Design and Manufacturing*. John Wiley & Sons, 1996.
- J. Soto. Statistical testing of random numbers. In *Proceedings of the 22nd National Information Systems Security Conference*, 1999.
- K. E. Stecke. Machine interference: assignment of machines to operators. In G. Salvendy, editor, *Handbook of Industrial Engineering*, chapter 57. John Wiley & Sons, 1992.
- N. M. Steiger and J. R. Wilson. An improved batch means procedure for simulation output analysis. *Management Science*, 48(12):1569–1586, 2002.
- H. C. Tijms. *A First Course in Stochastic Models*. John Wiley & Sons, 2003.
- P. D. Welch. A graphical approach to the initial transient problem in steady state simulation. In *10th IMACS World Congress on System Simulation and Scientific Computation*, pages 219–221, 1983a.
- P. D. Welch. The statistical analysis of simulation results. In S. Lavenberg, editor, *Computer Performance Modeling Handbook*, pages 267–329. Elsevier, 1983b.

- K. P. White, M. J. Cobb, and S. C. Spratt. A comparison of five steady-state truncation heuristics for simulation. In Proceeding of the 2000 Winter Simulation Conference, 2000.
- W. Whitt. The queueing network analyzer. *The Bell System Technical Journal*, 62(9):2779–2815, 1983.
- W. Whitt. Planning queueing simulations. *Management Science*, 35(11):1341–1366, 1989.
- W. Whitt. Approximations for the GI/G/m queue. *Productions and Operations Management*, 2(2):114–161, 1993.
- J. R. Wilson and A. A. B. Pritsker. A survey of research on the simulation startup problem. *Simulation*, 31(2):55–58, 1978.
- P. H. Zipkin. *Foundations of Inventory Management*. McGraw-Hill, 2000.



---

## APPENDIX A

---

# COMMON DISTRIBUTIONS

### A.1 DISCRETE DISTRIBUTIONS

TABLE A 1

Bernoulli	$\text{Ber}(p)$
Parameters:	$0 < p < 1$ , probability of success
PMF:	$P\{X = 1\} = p$ $P\{X = 0\} = 1 - p$
Expected value:	$E[X] = p$
Variance:	$\text{Var}[X] = p(1 - p)$
Arena™ generation:	<code>DISC(p, 1, 1.0, 0[,Stream])</code>
Spreadsheet generation:	<code>=IF(RAND()&lt; p, 1, 0)</code>
Modeling:	The number of successes in one trial

TABLE A 2

Binomial	$\text{Binom}(n, p)$
Parameters:	$0 < p < 1$ , probability of success, $n$ , number of trials
PMF:	$P\{X = x\} = \binom{n}{x} p^x (1 - p)^{n-x}, \quad x = 0, 1, \dots, n$
Expected value:	$E[X] = np$
Variance:	$\text{Var}[X] = np(1 - p)$
Arena™ generation:	Not available, use convolution of Bernoulli
Spreadsheet generation:	<code>=BINOM.INV(n,p,RAND())</code>
Modeling:	The number of successes in $n$ trials

**TABLE A 3**

Shifted Geometric	Shifted Geo( $p$ )
Parameters:	$0 < p < 1$ , probability of success
PMF:	$P\{X = x\} = p(1 - p)^{x-1}$ $x = 1, 2, \dots$ ,
Expected value:	$E[X] = 1/p$
Variance:	$\text{Var}[X] = (1 - p)/p^2$
Arena™ generation:	$1 + \text{AINT}(\text{LN}(1-\text{UNIF}(0,1))/\text{LN}(1-p))$
Spreadsheet generation:	$= 1 + \text{INT}(\text{LN}(1-\text{RAND})/\text{LN}(1-p))$
Modeling:	The number of trials until the first success

**TABLE A 4**

Negative Binomial Definition. 1	NB1( $r,p$ )
Parameters:	$0 < p < 1$ , probability of success, $r$ th success
PMF:	$P\{X = x\} = \binom{x-1}{r-1} p^r (1-p)^{x-r}$ , $x = r, r+1, \dots$ ,
Expected value:	$E[X] = 1/p$
Variance:	$\text{Var}[X] = (1 - p)/p^2$
Arena™ generation:	Not available, count until $r$ th success
Spreadsheet generation:	Not available, count until $r$ th success
Modeling:	The number of trials until the $r$ th success

**TABLE A 5**

Negative Binomial Definition. 2	NB2( $r,p$ )
Parameters:	$0 < p < 1$ , probability of success, $r$ th success
PMF:	$P\{Y = y\} = \binom{y+r-1}{r-1} p^r (1-p)^y$ , $y = 0, 1, \dots$
Expected value:	$E[Y] = r(1 - p)/p$
Variance:	$\text{Var}[Y] = r(1 - p)/p^2$
Arena™ generation:	Not available, count until $r$ th success
Spreadsheet generation:	Not available, count until $r$ th success
Modeling:	The number of failures prior to the $r$ th success

**TABLE A 6**

Poisson	Pois( $\lambda$ )
Parameters:	$\lambda > 0$
PMF:	$P\{X = x\} = \frac{e^{-\lambda} \lambda^x}{x!}$ , $x = 0, 1, \dots$
Expected value:	$E[X] = \lambda$
Variance:	$\text{Var}[X] = \lambda$
Arena™ generation:	POIS( $\lambda$ ,[Stream])
Spreadsheet generation:	Not available, approximate with lookup table approach
Modeling:	The number of occurrences during a period of time

**TABLE A 7**

<b>Discrete Uniform</b>	$DU(a, b)$
Parameters:	$a \leq b$
PMF:	$P\{X = x\} = \frac{1}{b-a+1}, \quad x = a, a+1, \dots, b$
Expected value:	$E[X] = (b+a)/2$
Variance:	$\text{Var}[X] = ((b-a+1)^2 - 1)/12$
Arena™ generation:	<code>AINT(UNIF(a, b) + 0.5)</code>
Spreadsheet generation:	<code>=RANDBETWEEN(a, b)</code>
Modeling:	Equal occurrence over a range of integers

**A.2 CONTINUOUS DISTRIBUTIONS****TABLE A 8**

<b>Uniform</b>	$U(a, b)$
Parameters:	$a = \text{minimum}, b = \text{maximum}, -\infty < a < b < \infty$
PDF:	$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$
CDF:	$F(x) = \begin{cases} 0 & \text{if } x < a \\ \frac{x-a}{b-a} & \text{if } a \leq x \leq b, \\ 1 & \text{if } b > x \end{cases}$
Inverse CDF:	$F^{-1}(p) = a + p(b-a) \text{ if } 0 < p < 1$
Expected value:	$E[X] = \frac{a+b}{2}$
Variance:	$V[X] = \frac{(b-a)^2}{12}$
Arena™ generation:	<code>UNIF(Min,Max[,Stream])</code>
Spreadsheet generation:	<code>=a + RAND()*(b-a)</code>
Modeling:	Assumes equally likely across the range, when you have lack of data, task times

**TABLE A 9**

<b>Normal</b>	$N(\mu, \sigma^2)$
Parameters:	$-\infty < \mu < +\infty$ (mean), $\sigma^2 > 0$ (variance)
CDF:	No closed form
Inverse CDF:	No closed form
Expected value:	$E[X] = \mu$
Variance:	$Var[X] = \sigma^2$
Arena™ generation:	<code>NORM(<math>\mu, \sigma</math>[,Stream])</code>
Spreadsheet generation:	<code>=NORM.INV(RAND(), <math>\mu, \sigma</math>)</code>
Modeling:	Task times, errors

**TABLE A 10**

<b>Exponential</b>	$EXPO(1/\lambda)$
Parameters:	$\lambda > 0$
PDF:	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ \lambda e^{-\lambda x} & \text{if } x \geq 0 \end{cases}$
CDF:	$F(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 - e^{-\lambda x} & \text{if } x \geq 0 \end{cases}$
Inverse CDF:	$F^{-1}(p) = (-1/\lambda) \ln (1 - p)$ if $0 < p < 1$
Expected value:	$E[X] = 1/\lambda$
Variance:	$Var[X] = 1/\lambda^2$
Arena™ generation:	<code>EXPO(mean[,Stream])</code>
Spreadsheet generation:	<code>=(-1/\lambda)LN(1-RAND())</code>
Modeling:	Time between arrivals, time to failure, highly variable task time

**TABLE A 11**

<b>Weibull</b>	$WEIB(\beta, \alpha)$
Parameters:	$\beta > 0$ (scale), $\alpha > 0$ (shape)
CDF:	$F(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 - e^{(-x/\beta)^\alpha} & \text{if } x \geq 0 \end{cases}$
Inverse CDF:	$F^{-1}(p) = \beta[-\ln (1 - p)]^{1/\alpha}$ if $0 < p < 1$
Expected value:	$E[X] = \left(\frac{\beta}{\alpha}\right) \Gamma\left(\frac{1}{\alpha}\right)$
Variance:	$Var[X] = \left(\frac{\beta^2}{\alpha}\right) \{2\Gamma\left(\frac{2}{\alpha}\right) - \left(\frac{1}{\alpha}\right) (\Gamma\left(\frac{1}{\alpha}\right))^2\}$
Arena™ generation:	<code>WEIB(scale, shape[,Stream])</code>
Spreadsheet generation:	<code>=(<math>\beta</math>)(-LN(1 - RAND()) <math>\wedge</math> (1/<math>\alpha</math>))</code>
Modeling:	Task times, time to failure

**TABLE A 12**

Erlang	$\text{Erlang}(r, \beta)$
Parameters:	$r > 0$ , integer, $\beta > 0$ (scale)
CDF:	$F(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 - e^{(-x/\beta)} \sum_{j=0}^{r-1} \frac{(x/\beta)^j}{j!} & \text{if } x \geq 0 \end{cases}$
Inverse CDF:	No closed form
Expected value:	$E[X] = r\beta$
Variance:	$\text{Var}[X] = r\beta^2$
Arena™ generation:	ERLA(E[X], r[,Stream])
Spreadsheet generation:	= GAMMA.INV(RAND(), r, β)
Modeling:	Task times, lead time, time to failure,

**TABLE A 13**

Gamma	$\text{Gamma}(\alpha, \beta)$
Parameters:	$\alpha > 0$ , shape, $\beta > 0$ (scale)
CDF:	No closed form
Inverse CDF:	No closed form
Expected Value:	$E[X] = \alpha\beta$
Variance:	$\text{Var}[X] = \alpha\beta^2$
Arena™ generation:	GAMM(β, α[,Stream])
Spreadsheet generation:	= GAMMA.INV(RAND(), α, β)
Modeling:	Task times, lead time, time to failure,

**TABLE A 14**

Triangular	$\text{TRIA}(a, m, b)$
Parameters:	$a = \text{minimum}$ , $m = \text{mode}$ , $b = \text{maximum}$
CDF:	$F(x) = \begin{cases} 0 & x < a \\ \frac{(x-a)^2}{(b-a)(m-a)} & a \leq x \leq m \\ 1 - \frac{(b-x)^2}{(b-a)(b-m)} & m < x \leq b \\ 1 & b < x \end{cases}$
Inverse CDF:	$F^{-1}(p) = \begin{cases} a + \sqrt{(b-a)(m-a)p} & 0 < p < \frac{m-a}{b-a} \\ b - \sqrt{(b-a)(b-m)(1-p)} & \frac{m-a}{b-a} \leq p \leq 1 \end{cases}$
Expected value:	$E[X] = (a + m + b)/3$
Variance:	$\text{Var}[X] = \frac{a^2+b^2+m^2-ab-am-bm}{18}$
Arena™ generation:	TRIA(min, mode, max[,Stream])
Spreadsheet generation:	Implement $F^{-1}(p)$ as VBA function
Modeling:	Task times, activity time when data is limited

**TABLE A 15**

<b>Beta</b>	BETA( $\alpha_1, \alpha_2$ )
Parameters:	Shape parameters $\alpha_1 > 0, \alpha_2 > 0$
CDF:	No closed form
Inverse CDF:	No closed form
Expected value:	$E[X] = \frac{\alpha_1}{\alpha_1 + \alpha_2}$
Variance:	$Var[X] = \frac{\alpha_1 \alpha_2}{(\alpha_1 + \alpha_2)^2 (\alpha_1 + \alpha_2 + 1)}$
Arena™ generation:	BETA( $\alpha_1, \alpha_2$ ,[Stream])
Spreadsheet generation:	BETA.INV(RAND(), $\alpha_1, \alpha_2$ )
Modeling:	Activity time when data is limited, probabilities

**TABLE A 16**

<b>Lognormal</b>	LOGN( $\mu_l, \sigma_l$ )
Parameters:	$\mu = \ln (\mu_l / \sqrt{\sigma_l^2 + \mu_l^2})$ $\sigma^2 = \ln ((\sigma_l^2 / \mu_l^2) + 1)$
CDF:	No closed form
Inverse CDF:	No closed form
Expected value:	$E[X] = \mu_l = e^{\mu + \sigma^2/2}$
Variance:	$Var[X] = \sigma_l^2 = e^{2\mu + \sigma^2} (e^{\sigma^2} - 1)$
Arena™ generation:	LOGN( $\mu_l, \sigma_l$ ,[Stream])
Spreadsheet generation:	LOGNORM.INV(RAND(), $\mu, \sigma$ )
Modeling:	Task times, time to failure

---

## **APPENDIX B**

---

### **STATISTICAL TABLES**

**TABLE B 1 Cumulative Standard Normal Distribution**

<i>z</i>	-0.09	-0.08	-0.07	-0.06	-0.05
-3.9	0.000033	0.000034	0.000036	0.000037	0.000039
-3.8	0.000050	0.000052	0.000054	0.000057	0.000059
-3.7	0.000075	0.000078	0.000082	0.000085	0.000088
-3.6	0.000112	0.000117	0.000121	0.000126	0.000131
-3.5	0.000165	0.000172	0.000178	0.000185	0.000193
-3.4	0.000242	0.000251	0.000260	0.000270	0.000280
-3.3	0.000349	0.000362	0.000376	0.000390	0.000404
-3.2	0.000501	0.000519	0.000538	0.000557	0.000577
-3.1	0.000711	0.000736	0.000762	0.000789	0.000816
-3	0.001001	0.001035	0.001070	0.001107	0.001144
-2.9	0.001395	0.001441	0.001489	0.001538	0.001589
-2.8	0.001926	0.001988	0.002052	0.002118	0.002186
-2.7	0.002635	0.002718	0.002803	0.002890	0.002980
-2.6	0.003573	0.003681	0.003793	0.003907	0.004025
-2.5	0.004799	0.004940	0.005085	0.005234	0.005386
-2.4	0.006387	0.006569	0.006756	0.006947	0.007143
-2.3	0.008424	0.008656	0.008894	0.009137	0.009387
-2.2	0.011011	0.011304	0.011604	0.011911	0.012224
-2.1	0.014262	0.014629	0.015003	0.015386	0.015778
-2	0.018309	0.018763	0.019226	0.019699	0.020182
-1.9	0.023295	0.023852	0.024419	0.024998	0.025588
-1.8	0.029379	0.030054	0.030742	0.031443	0.032157
-1.7	0.036727	0.037538	0.038364	0.039204	0.040059
-1.6	0.045514	0.046479	0.047460	0.048457	0.049471
-1.5	0.055917	0.057053	0.058208	0.059380	0.060571
-1.4	0.068112	0.069437	0.070781	0.072145	0.073529
-1.3	0.082264	0.083793	0.085343	0.086915	0.088508
-1.2	0.098525	0.100273	0.102042	0.103835	0.105650
-1.1	0.117023	0.119000	0.121000	0.123024	0.125072
-1	0.137857	0.140071	0.142310	0.144572	0.146859
-0.9	0.161087	0.163543	0.166023	0.168528	0.171056
-0.8	0.186733	0.189430	0.192150	0.194895	0.197663
-0.7	0.214764	0.217695	0.220650	0.223627	0.226627
-0.6	0.245097	0.248252	0.251429	0.254627	0.257846
-0.5	0.277595	0.280957	0.284339	0.287740	0.291160
-0.4	0.312067	0.315614	0.319178	0.322758	0.326355
-0.3	0.348268	0.351973	0.355691	0.359424	0.363169
-0.2	0.385908	0.389739	0.393580	0.397432	0.401294
-0.1	0.424655	0.428576	0.432505	0.436441	0.440382
0	0.464144	0.468119	0.472097	0.476078	0.480061

**TABLE B 2 Cumulative Standard Normal Distribution**

<i>z</i>	-0.04	-0.03	-0.02	-0.01	0
-3.9	0.000041	0.000042	0.000044	0.000046	0.000048
-3.8	0.000062	0.000064	0.000067	0.000069	0.000072
-3.7	0.000092	0.000096	0.000100	0.000104	0.000108
-3.6	0.000136	0.000142	0.000147	0.000153	0.000159
-3.5	0.000200	0.000208	0.000216	0.000224	0.000233
-3.4	0.000291	0.000302	0.000313	0.000325	0.000337
-3.3	0.000419	0.000434	0.000450	0.000466	0.000483
-3.2	0.000598	0.000619	0.000641	0.000664	0.000687
-3.1	0.000845	0.000874	0.000904	0.000935	0.000968
-3	0.001183	0.001223	0.001264	0.001306	0.001350
-2.9	0.001641	0.001695	0.001750	0.001807	0.001866
-2.8	0.002256	0.002327	0.002401	0.002477	0.002555
-2.7	0.003072	0.003167	0.003264	0.003364	0.003467
-2.6	0.004145	0.004269	0.004396	0.004527	0.004661
-2.5	0.005543	0.005703	0.005868	0.006037	0.006210
-2.4	0.007344	0.007549	0.007760	0.007976	0.008198
-2.3	0.009642	0.009903	0.010170	0.010444	0.010724
-2.2	0.012545	0.012874	0.013209	0.013553	0.013903
-2.1	0.016177	0.016586	0.017003	0.017429	0.017864
-2	0.020675	0.021178	0.021692	0.022216	0.022750
-1.9	0.026190	0.026803	0.027429	0.028067	0.028717
-1.8	0.032884	0.033625	0.034380	0.035148	0.035930
-1.7	0.040930	0.041815	0.042716	0.043633	0.044565
-1.6	0.050503	0.051551	0.052616	0.053699	0.054799
-1.5	0.061780	0.063008	0.064255	0.065522	0.066807
-1.4	0.074934	0.076359	0.077804	0.079270	0.080757
-1.3	0.090123	0.091759	0.093418	0.095098	0.096800
-1.2	0.107488	0.109349	0.111232	0.113139	0.115070
-1.1	0.127143	0.129238	0.131357	0.133500	0.135666
-1	0.149170	0.151505	0.153864	0.156248	0.158655
-0.9	0.173609	0.176186	0.178786	0.181411	0.184060
-0.8	0.200454	0.203269	0.206108	0.208970	0.211855
-0.7	0.229650	0.232695	0.235762	0.238852	0.241964
-0.6	0.261086	0.264347	0.267629	0.270931	0.274253
-0.5	0.294599	0.298056	0.301532	0.305026	0.308538
-0.4	0.329969	0.333598	0.337243	0.340903	0.344578
-0.3	0.366928	0.370700	0.374484	0.378280	0.382089
-0.2	0.405165	0.409046	0.412936	0.416834	0.420740
-0.1	0.444330	0.448283	0.452242	0.456205	0.460172
0	0.484047	0.488034	0.492022	0.496011	0.500000

**TABLE B 3 Cumulative Standard Normal Distribution**

<i>z</i>	0	0.01	0.02	0.03	0.04
0	0.500000	0.503989	0.507978	0.511966	0.515953
0.1	0.539828	0.543795	0.547758	0.551717	0.555670
0.2	0.579260	0.583166	0.587064	0.590954	0.594835
0.3	0.617911	0.621720	0.625516	0.629300	0.633072
0.4	0.655422	0.659097	0.662757	0.666402	0.670031
0.5	0.691462	0.694974	0.698468	0.701944	0.705401
0.6	0.725747	0.729069	0.732371	0.735653	0.738914
0.7	0.758036	0.761148	0.764238	0.767305	0.770350
0.8	0.788145	0.791030	0.793892	0.796731	0.799546
0.9	0.815940	0.818589	0.821214	0.823814	0.826391
1	0.841345	0.843752	0.846136	0.848495	0.850830
1.1	0.864334	0.866500	0.868643	0.870762	0.872857
1.2	0.884930	0.886861	0.888768	0.890651	0.892512
1.3	0.903200	0.904902	0.906582	0.908241	0.909877
1.4	0.919243	0.920730	0.922196	0.923641	0.925066
1.5	0.933193	0.934478	0.935745	0.936992	0.938220
1.6	0.945201	0.946301	0.947384	0.948449	0.949497
1.7	0.955435	0.956367	0.957284	0.958185	0.959070
1.8	0.964070	0.964852	0.965620	0.966375	0.967116
1.9	0.971283	0.971933	0.972571	0.973197	0.973810
2	0.977250	0.977784	0.978308	0.978822	0.979325
2.1	0.982136	0.982571	0.982997	0.983414	0.983823
2.2	0.986097	0.986447	0.986791	0.987126	0.987455
2.3	0.989276	0.989556	0.989830	0.990097	0.990358
2.4	0.991802	0.992024	0.992240	0.992451	0.992656
2.5	0.993790	0.993963	0.994132	0.994297	0.994457
2.6	0.995339	0.995473	0.995604	0.995731	0.995855
2.7	0.996533	0.996636	0.996736	0.996833	0.996928
2.8	0.997445	0.997523	0.997599	0.997673	0.997744
2.9	0.998134	0.998193	0.998250	0.998305	0.998359
3	0.998650	0.998694	0.998736	0.998777	0.998817
3.1	0.999032	0.999065	0.999096	0.999126	0.999155
3.2	0.999313	0.999336	0.999359	0.999381	0.999402
3.3	0.999517	0.999534	0.999550	0.999566	0.999581
3.4	0.999663	0.999675	0.999687	0.999698	0.999709
3.5	0.999767	0.999776	0.999784	0.999792	0.999800
3.6	0.999841	0.999847	0.999853	0.999858	0.999864
3.7	0.999892	0.999896	0.999900	0.999904	0.999908
3.8	0.999928	0.999931	0.999933	0.999936	0.999938
3.9	0.999952	0.999954	0.999956	0.999958	0.999959

**TABLE B 4 Cumulative Standard Normal Distribution**

<i>z</i>	0.05	0.06	0.07	0.08	0.09
0	0.519939	0.523922	0.527903	0.531881	0.535856
0.1	0.559618	0.563559	0.567495	0.571424	0.575345
0.2	0.598706	0.602568	0.606420	0.610261	0.614092
0.3	0.636831	0.640576	0.644309	0.648027	0.651732
0.4	0.673645	0.677242	0.680822	0.684386	0.687933
0.5	0.708840	0.712260	0.715661	0.719043	0.722405
0.6	0.742154	0.745373	0.748571	0.751748	0.754903
0.7	0.773373	0.776373	0.779350	0.782305	0.785236
0.8	0.802337	0.805105	0.807850	0.810570	0.813267
0.9	0.828944	0.831472	0.833977	0.836457	0.838913
1	0.853141	0.855428	0.857690	0.859929	0.862143
1.1	0.874928	0.876976	0.879000	0.881000	0.882977
1.2	0.894350	0.896165	0.897958	0.899727	0.901475
1.3	0.911492	0.913085	0.914657	0.916207	0.917736
1.4	0.926471	0.927855	0.929219	0.930563	0.931888
1.5	0.939429	0.940620	0.941792	0.942947	0.944083
1.6	0.950529	0.951543	0.952540	0.953521	0.954486
1.7	0.959941	0.960796	0.961636	0.962462	0.963273
1.8	0.967843	0.968557	0.969258	0.969946	0.970621
1.9	0.974412	0.975002	0.975581	0.976148	0.976705
2	0.979818	0.980301	0.980774	0.981237	0.981691
2.1	0.984222	0.984614	0.984997	0.985371	0.985738
2.2	0.987776	0.988089	0.988396	0.988696	0.988989
2.3	0.990613	0.990863	0.991106	0.991344	0.991576
2.4	0.992857	0.993053	0.993244	0.993431	0.993613
2.5	0.994614	0.994766	0.994915	0.995060	0.995201
2.6	0.995975	0.996093	0.996207	0.996319	0.996427
2.7	0.997020	0.997110	0.997197	0.997282	0.997365
2.8	0.997814	0.997882	0.997948	0.998012	0.998074
2.9	0.998411	0.998462	0.998511	0.998559	0.998605
3	0.998856	0.998893	0.998930	0.998965	0.998999
3.1	0.999184	0.999211	0.999238	0.999264	0.999289
3.2	0.999423	0.999443	0.999462	0.999481	0.999499
3.3	0.999596	0.999610	0.999624	0.999638	0.999651
3.4	0.999720	0.999730	0.999740	0.999749	0.999758
3.5	0.999807	0.999815	0.999822	0.999828	0.999835
3.6	0.999869	0.999874	0.999879	0.999883	0.999888
3.7	0.999912	0.999915	0.999918	0.999922	0.999925
3.8	0.999941	0.999943	0.999946	0.999948	0.999950
3.9	0.999961	0.999963	0.999964	0.999966	0.999967

**TABLE B 5 Percentage Points  $t_{v,\alpha}$  of the Student- $t$  Distribution**

$v \backslash \alpha$	0.1	0.05	0.025	0.01	0.005	0.0025	0.001	0.0005
1	3.078	6.314	12.706	31.821	63.657	127.321	318.309	636.619
2	1.886	2.920	4.303	6.965	9.925	14.089	22.327	31.599
3	1.638	2.353	3.182	4.541	5.841	7.453	10.215	12.924
4	1.533	2.132	2.776	3.747	4.604	5.598	7.173	8.610
5	1.476	2.015	2.571	3.365	4.032	4.773	5.893	6.869
6	1.440	1.943	2.447	3.143	3.707	4.317	5.208	5.959
7	1.415	1.895	2.365	2.998	3.499	4.029	4.785	5.408
8	1.397	1.860	2.306	2.896	3.355	3.833	4.501	5.041
9	1.383	1.833	2.262	2.821	3.250	3.690	4.297	4.781
10	1.372	1.812	2.228	2.764	3.169	3.581	4.144	4.587
11	1.363	1.796	2.201	2.718	3.106	3.497	4.025	4.437
12	1.356	1.782	2.179	2.681	3.055	3.428	3.930	4.318
13	1.350	1.771	2.160	2.650	3.012	3.372	3.852	4.221
14	1.345	1.761	2.145	2.624	2.977	3.326	3.787	4.140
15	1.341	1.753	2.131	2.602	2.947	3.286	3.733	4.073
16	1.337	1.746	2.120	2.583	2.921	3.252	3.686	4.015
17	1.333	1.740	2.110	2.567	2.898	3.222	3.646	3.965
18	1.330	1.734	2.101	2.552	2.878	3.197	3.610	3.922
19	1.328	1.729	2.093	2.539	2.861	3.174	3.579	3.883
20	1.325	1.725	2.086	2.528	2.845	3.153	3.552	3.850
21	1.323	1.721	2.080	2.518	2.831	3.135	3.527	3.819
22	1.321	1.717	2.074	2.508	2.819	3.119	3.505	3.792
23	1.319	1.714	2.069	2.500	2.807	3.104	3.485	3.768
24	1.318	1.711	2.064	2.492	2.797	3.091	3.467	3.745
25	1.316	1.708	2.060	2.485	2.787	3.078	3.450	3.725
26	1.315	1.706	2.056	2.479	2.779	3.067	3.435	3.707
27	1.314	1.703	2.052	2.473	2.771	3.057	3.421	3.690
28	1.313	1.701	2.048	2.467	2.763	3.047	3.408	3.674
29	1.311	1.699	2.045	2.462	2.756	3.038	3.396	3.659
30	1.310	1.697	2.042	2.457	2.750	3.030	3.385	3.646
31	1.309	1.696	2.040	2.453	2.744	3.022	3.375	3.633
32	1.309	1.694	2.037	2.449	2.738	3.015	3.365	3.622
33	1.308	1.692	2.035	2.445	2.733	3.008	3.356	3.611
34	1.307	1.691	2.032	2.441	2.728	3.002	3.348	3.601
35	1.306	1.690	2.030	2.438	2.724	2.996	3.340	3.591
40	1.303	1.684	2.021	2.423	2.704	2.971	3.307	3.551
45	1.301	1.679	2.014	2.412	2.690	2.952	3.281	3.520
50	1.299	1.676	2.009	2.403	2.678	2.937	3.261	3.496
$\infty$	1.282	1.645	1.960	2.326	2.576	2.807	3.090	3.291

**TABLE B 6 Percentage Points  $\chi^2_{v,\alpha}$  of the Chi-Square Distribution ( $v$ ) degrees of freedom**

$v \backslash \alpha$	0.1	0.05	0.025	0.01	0.005	0.0025	0.001	0.0005
1	2.706	3.841	5.024	6.635	7.879	9.141	10.828	12.116
2	4.605	5.991	7.378	9.210	10.597	11.983	13.816	15.202
3	6.251	7.815	9.348	11.345	12.838	14.320	16.266	17.730
4	7.779	9.488	11.143	13.277	14.860	16.424	18.467	19.997
5	9.236	11.070	12.833	15.086	16.750	18.386	20.515	22.105
6	10.645	12.592	14.449	16.812	18.548	20.249	22.458	24.103
7	12.017	14.067	16.013	18.475	20.278	22.040	24.322	26.018
8	13.362	15.507	17.535	20.090	21.955	23.774	26.124	27.868
9	14.684	16.919	19.023	21.666	23.589	25.462	27.877	29.666
10	15.987	18.307	20.483	23.209	25.188	27.112	29.588	31.420
11	17.275	19.675	21.920	24.725	26.757	28.729	31.264	33.137
12	18.549	21.026	23.337	26.217	28.300	30.318	32.909	34.821
13	19.812	22.362	24.736	27.688	29.819	31.883	34.528	36.478
14	21.064	23.685	26.119	29.141	31.319	33.426	36.123	38.109
15	22.307	24.996	27.488	30.578	32.801	34.950	37.697	39.719
16	23.542	26.296	28.845	32.000	34.267	36.456	39.252	41.308
17	24.769	27.587	30.191	33.409	35.718	37.946	40.790	42.879
18	25.989	28.869	31.526	34.805	37.156	39.422	42.312	44.434
19	27.204	30.144	32.852	36.191	38.582	40.885	43.820	45.973
20	28.412	31.410	34.170	37.566	39.997	42.336	45.315	47.498
21	29.615	32.671	35.479	38.932	41.401	43.775	46.797	49.011
22	30.813	33.924	36.781	40.289	42.796	45.204	48.268	50.511
23	32.007	35.172	38.076	41.638	44.181	46.623	49.728	52.000
24	33.196	36.415	39.364	42.980	45.559	48.034	51.179	53.479
25	34.382	37.652	40.646	44.314	46.928	49.435	52.620	54.947
26	35.563	38.885	41.923	45.642	48.290	50.829	54.052	56.407
27	36.741	40.113	43.195	46.963	49.645	52.215	55.476	57.858
28	37.916	41.337	44.461	48.278	50.993	53.594	56.892	59.300
29	39.087	42.557	45.722	49.588	52.336	54.967	58.301	60.735
30	40.256	43.773	46.979	50.892	53.672	56.332	59.703	62.162
31	41.422	44.985	48.232	52.191	55.003	57.692	61.098	63.582
32	42.585	46.194	49.480	53.486	56.328	59.046	62.487	64.995
33	43.745	47.400	50.725	54.776	57.648	60.395	63.870	66.403
34	44.903	48.602	51.966	56.061	58.964	61.738	65.247	67.803
35	46.059	49.802	53.203	57.342	60.275	63.076	66.619	69.199
40	51.805	55.758	59.342	63.691	66.766	69.699	73.402	76.095
50	63.167	67.505	71.420	76.154	79.490	82.664	86.661	89.561
60	74.397	79.082	83.298	88.379	91.952	95.344	99.607	102.695
70	85.527	90.531	95.023	100.425	104.215	107.808	112.317	115.578
80	96.578	101.879	106.629	112.329	116.321	120.102	124.839	128.261
90	107.565	113.145	118.136	124.116	128.299	132.256	137.208	140.782
100	118.498	124.342	129.561	135.807	140.169	144.293	149.449	153.167

**TABLE B 7 Kolmogorov–Smirnov Test Critical Values**

<i>n</i>	$D_{0.1}$	$D_{0.05}$	$D_{0.01}$
10	0.36866	0.40925	0.48893
11	0.35242	0.39122	0.46770
12	0.33815	0.37543	0.44905
13	0.32549	0.36143	0.43247
14	0.31417	0.34890	0.41762
15	0.30397	0.33760	0.40420
16	0.29472	0.32733	0.39201
17	0.28627	0.31796	0.38086
18	0.27851	0.30936	0.37062
19	0.27136	0.30143	0.36117
20	0.26473	0.29408	0.35241
25	0.23768	0.26404	0.31657
30	0.21756	0.24170	0.28987
35	0.20185	0.22425	0.26897
Over 35	$1.22/\sqrt{n}$	$1.36/\sqrt{n}$	$1.63/\sqrt{n}$

See Miller [1956]. Additional values can be computed at: <http://www.ciphersbyritter.com/JAVASCRP/NORMCHIK.HTM#KolSmir>

---

## APPENDIX C

---

### DISTRIBUTIONS, OPERATORS, FUNCTIONS IN ARENA

TABLE C 1 Arena's Distributions

Beta	BETA(Alpha,Alpha2[,Stream])
Normal	NORM(Mean,SD[,Stream])
Empirical continuous	CONT(Prob1,Value1,Prob2,Value2, ... [,Stream])
NSExo	Nonhomogeneous Poisson process
Empirical discrete	DISC(Prob1,Value1,Prob2,Value2, ... [,Stream])
Poisson	POIS(Mean[,Stream])
K-Erlang	ERLA(Mean,k[,Stream])
Lognormal	LOGN(LogMean,LogStd[,Stream])
Uniform(0,1)	RA
Exponential	EXPO(Mean[,Stream])
Triangular	TRIA(Min,Mode,Max[,Stream])
Gamma	GAMM(scale,shape[,Stream])
Uniform	UNIF(Min,Max[,Stream])
Johnson	JOHN(shape1,shape2,scale,location[,Stream])
Weibull	WEIB(scale,shape[,Stream])

---

**TABLE C 2 Mathematical and Logical Operators**

Operator	Operation	Priority
<b>Math operators</b>		
**	Exponentiation	1(highest)
/	Division	2
*	Multiplication	2
-	Subtraction	3
+	Addition	3
<b>Logical operators</b>		
.EQ., ==	Equality comparison	4
.NE., <>	Nonequality comparison	4
.LT., <	Less than comparison	4
.GT., >	Greater than comparison	4
.LT., <=	Less than or equal to comparison	4
.GE., >=	Greater than or equal to comparison	4
.AND., &&	Conjunction (and)	5
.OR.,	Inclusive disjunction (or)	5

**TABLE C 3 Arena's Mathematical Functions**

Function	Description
ABS( <i>a</i> )	Absolute value
ACOS( <i>a</i> )	Arc cosine
AINT( <i>a</i> )	Truncate
AMOD( <i>a</i> <sub>1</sub> , <i>a</i> <sub>2</sub> )	Real remainder, returns ( <i>a</i> <sub>1</sub> - (AINT( $\frac{a_1}{a_2}$ ) × <i>a</i> <sub>2</sub> ))
ANINT( <i>a</i> )	Round to nearest integer
ASIN( <i>a</i> )	Arc sine
ATAN( <i>a</i> )	Arc tangent
COS( <i>a</i> )	Cosine
EP( <i>a</i> )	Exponential( $e^a$ )
HCOS( <i>a</i> )	Hyperbolic cosine
HSIN( <i>a</i> )	Hyperbolic sine
HTAN( <i>a</i> )	Hyperbolic tangent
MN( <i>a</i> <sub>1</sub> , <i>a</i> <sub>2</sub> , ...)	Minimum value
MOD( <i>a</i> <sub>1</sub> , <i>a</i> <sub>2</sub> )	same as AMOD(AINT( <i>a</i> <sub>1</sub> ), AINT( <i>a</i> <sub>2</sub> ))
MX( <i>a</i> <sub>1</sub> , <i>a</i> <sub>2</sub> , ...)	Maximum value
LN( <i>a</i> )	Natural logarithm
LOG( <i>a</i> )	Common logarithm
SIN( <i>a</i> )	Sine
SQRT( <i>a</i> )	Square root
TAN( <i>a</i> )	Tangent

---

## **APPENDIX D**

---

### **QUEUING THEORY FORMULAS**

**TABLE D 1****Results of  $P_0$  and  $P_n$  for M/M/1**

$$\begin{aligned}\lambda_n &= \lambda \\ \mu_n &= \mu \\ r &= \lambda/\mu\end{aligned}\quad P_0 = 1 - r \quad P_n = P_0 r^n$$

**Results of  $P_0$  and  $P_n$  for M/M/c**

$$\begin{aligned}\lambda_n &= \lambda \\ \mu_n &= \begin{cases} n\mu & 0 \leq n < c \\ c\mu & n \geq c \end{cases} \\ \rho &= \lambda/c\mu \quad r = \lambda/\mu\end{aligned}\quad P_0 = \left[ \sum_{n=0}^{c-1} \frac{r^n}{n!} + \frac{r^c}{c!(1-\rho)} \right]^{-1} \quad P_n = \begin{cases} \frac{(r^n)^2}{n!} P_0 & 1 \leq n < c \\ \frac{r^n}{c!c^{n-c}} P_0 & n \geq c \end{cases}$$

**Results of  $P_0$  and  $P_n$  for M/M/1/k**

$$\begin{aligned}\lambda_n &= \begin{cases} \lambda & n < k \\ 0 & n \geq k \end{cases} \\ \mu_n &= \begin{cases} \mu & 0 \leq n \leq k \\ 0 & n > k \end{cases} \\ \rho &= \lambda/c\mu \quad r = \lambda/\mu \\ \lambda_e &= \lambda(1 - P_k)\end{aligned}\quad P_0 = \begin{cases} \frac{1-r}{k+1} r^{k+1} & r \neq 1 \\ \frac{1}{k+1} & r = 1 \end{cases} \quad P_n = \begin{cases} P_0 r^n & r \neq 1 \\ \frac{1}{k+1} & r = 1 \end{cases}$$

**Results of  $P_0$  and  $P_n$  for M/M/c/k**

$$\begin{aligned}\lambda_n &= \begin{cases} \lambda & n < k \\ 0 & n \geq k \end{cases} \\ \mu_n &= \begin{cases} n\mu & 0 \leq n < c \\ c\mu & c \leq n \leq k \end{cases} \\ \rho &= \lambda/c\mu \quad r = \lambda/\mu \\ \lambda_e &= \lambda(1 - P_k)\end{aligned}\quad P_0 = \begin{cases} \left[ \sum_{n=0}^{c-1} \frac{r^n}{n!} + \frac{r^c}{c!} \frac{1 - \rho^{k-c+1}}{1 - \rho} \right]^{-1} & \rho \neq 1 \\ \left[ \sum_{n=0}^{c-1} \frac{r^n}{n!} + \frac{r^c}{c!} (k - c + 1) \right]^{-1} & \rho = 1 \end{cases} \quad P_n = \begin{cases} \frac{r^n}{n!} P_0 & 1 \leq n < c \\ \frac{r^n}{c!c^{n-c}} P_0 & c \leq n \leq k \end{cases}$$

**Results of  $P_0$  and  $P_n$  for M/G/c/c**

$$\begin{aligned}\lambda_n &= \begin{cases} \lambda & n < c \\ 0 & n \geq c \end{cases} \\ \mu_n &= \begin{cases} n\mu & 0 \leq n \leq c \\ 0 & n > c \end{cases} \\ \rho &= \lambda/c\mu \quad r = \lambda/\mu \\ \lambda_e &= \lambda(1 - P_k)\end{aligned}\quad P_0 = \left[ \sum_{n=0}^c \frac{r^n}{n!} \right]^{-1} \quad P_n = \frac{r^n}{n!} P_0 \quad 0 \leq n \leq c$$

**Results of  $P_0$  and  $P_n$  for M/M/1/k/k**

$$\begin{aligned}\lambda_n &= \begin{cases} (k-n)\lambda & 0 \leq n < k \\ 0 & n \geq k \end{cases} \\ \mu_n &= \begin{cases} (k-n)\lambda & 0 \leq n \leq k \\ 0 & n > k \end{cases} \\ r &= \lambda/\mu \quad \lambda_e = \lambda(k - L)\end{aligned}\quad P_0 = \left[ \sum_{n=0}^k \prod_{j=0}^{n-1} \left( \frac{\lambda_j}{\mu_{j+1}} \right) \right]^{-1} \quad P_n = \binom{k}{n} n! r^n P_0 \quad 0 \leq n \leq k$$

**Results of  $P_0$  and  $P_n$  for M/M/c/k/k**

$$\begin{aligned}\lambda_n &= \begin{cases} (k-n)\lambda & 0 \leq n < k \\ 0 & n \geq k \end{cases} \\ \mu_n &= \begin{cases} n\mu & 0 \leq n < c \\ c\mu & n \geq c \end{cases} \\ r &= \lambda/\mu \quad \lambda_e = \lambda(k - L)\end{aligned}\quad P_0 = \left[ \sum_{n=0}^k \prod_{j=0}^{n-1} \left( \frac{\lambda_j}{\mu_{j+1}} \right) \right]^{-1} \quad P_n = \begin{cases} \binom{k}{n} r^n P_0 & 1 \leq n < c \\ \binom{k}{n} \frac{n!}{c^{n-c} c!} r^n P_0 & c \leq n \leq k \end{cases}$$

**TABLE D 2 Results for  $L_q$  for Various Queueing Systems**

M/M/1	$L_q = \frac{r^2}{1 - r}$
M/M/c	$L_q = \left( \frac{r^c \rho}{c!(1 - \rho)^2} \right) P_0$
M/M/1/k	$L_q = \begin{cases} \frac{\rho}{1 - \rho} - \frac{\rho(k\rho^k + 1)}{1 - \rho^{k+1}} & \rho \neq 1 \\ \frac{k(k-1)}{2(k+1)} & \rho = 1 \end{cases}$
M/M/c/k	$L_q = \begin{cases} \frac{P_0 r^c \rho}{c!(1 - \rho)^2} [1 - \rho^{k-c} - (k-c)\rho^{k-c}(1-\rho)] & \rho < 1 \\ \frac{r^c (k-c)(k-c+1)}{2c!} P_0 & \rho = 1 \end{cases}$
M/G/c/c	$L_q = 0$
M/M/1/k/k	$L_q = k - \left( \frac{\lambda + \mu}{\lambda} \right) (1 - P_0)$
M/M/c/k/k	$L_q = \sum_{n=c}^k (n - c) P_n$

**TABLE D 3 Results M/M/c  $\rho = \lambda/c\mu$** 

$c$	$P_0$	$L_q$
1	$1 - \rho$	$\frac{\rho^2}{1 - \rho}$
2	$\frac{1 - \rho}{1 + \rho}$	$\frac{2\rho^3}{1 - \rho^2}$
3	$\frac{2(1 - \rho)}{2 + 4\rho + 3\rho^2}$	$\frac{9\rho^4}{2 + 2\rho - \rho^2 - 3\rho^3}$

**TABLE D 4 Results M/G/1 and M/D/1**

Model	Parameters	$L_q$
M/G/1	$E[ST] = \frac{1}{\mu}; \text{Var}[ST] = \sigma^2; r = \lambda/\mu$	$L_q = \frac{\lambda^2 \sigma^2 + r^2}{2(1 - r)}$
M/D/1	$E[ST] = \frac{1}{\mu}; \text{Var}[ST] = 0; r = \lambda/\mu$	$L_q = \frac{r^2}{2(1 - r)}$



---

## APPENDIX E

---

# INVENTORY THEORY FORMULAS

## ANALYTICAL RESULTS FOR (r,q) INVENTORY MODEL

- Let  $h$  be the holding cost per unit per time (\$/unit/time).
- Let  $b$  be the back order cost per unit per time (\$/unit/time).
- Let  $k$  be the cost per order (\$/order/time).
- Let  $r$  be the reorder point.
- Let  $Q$  be the reorder quantity.
- Let  $L$  be the constant lead time (measured in time-units).
- Let  $\lambda$  be the mean customer demand rate in units/time.
- Let  $g(x; t)$  be the probability mass function for Poisson distribution with rate  $\lambda$ , representing the number of events that occur in an interval of length

$$g(x; t) = P\{X(t) = x\} = \frac{e^{-\lambda t}(\lambda t)^x}{x!}$$

- Let  $G(x; t)$  be the cumulative distribution function for the Poisson distribution

$$G(x; t) = P\{X(t) \leq x\} = \sum_{i=0}^x g(i; t)$$

- Let  $G^0(x; t)$  be the complementary cumulative distribution function for the Poisson distribution

$$G^0(x; t) = 1 - G(x; t)$$

- Let  $G^1(x; t)$  be the first-order loss function for the Poisson distribution

$$G^1(x; t) = -(x - \lambda t)G^0(x; t) + (\lambda t)g(x; t)$$

- Let  $G^2(x; t)$  be the second-order loss function for the Poisson distribution

$$G^2(x; t) = \left(\frac{1}{2}\right) \left\{ [(x - \lambda t)^2 + x] G^0(x; t) - (\lambda t)(x - \lambda t)g(x; t) \right\}$$

- Let  $\overline{SO}$  be the steady-state expected fraction of time, there is no inventory on hand

$$\overline{SO} = \frac{1}{Q} \{ G^1(r; L) - G^1(r + Q; L) \}$$

- Let  $\overline{BO}$  be the steady-state expected amount of demand back ordered

$$\overline{BO} = \frac{1}{Q} \{ G^2(r; L) - G^2(r + Q; L) \}$$

- Let  $\overline{I}$  be the steady-state expected amount of inventory on hand

$$\overline{I} = \frac{1}{2}(Q + 1) + r - \lambda L + \overline{BO}$$

- Let  $\overline{OF}$  be the order frequency, orders/time

$$\overline{OF} = \frac{\lambda}{Q}$$

- Let  $\overline{TC}$  be the total cost per time (\$/time)

$$\overline{TC} = k\overline{OF} + h\overline{I} + b\overline{BO}$$

---

## **APPENDIX F**

---

### **USEFUL EQUATIONS**

$F(x) = P\{X \leq x\} = \int_{-\infty}^x f(u) du$	$\text{Var}[X] = \text{E}[X^2] - (\text{E}[X])^2$
$\text{Cov}[X, Y] = \text{E}[XY] - \text{E}[X]\text{E}[Y]$	$\text{Corr}[X, Y] = \frac{\text{Cov}[X, Y]}{\sqrt{\text{Var}[X]\text{Var}[Y]}}$
$\bar{X}(n) = \frac{1}{n} \sum_{i=1}^n X_i$	$\bar{Y}(n) = \frac{1}{t_n - t_0} \int_{t_0}^{t_n} Y(t) dt$
$S^2(n) = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$	$\bar{x} \pm t_{\alpha/2, n-1} \frac{s}{\sqrt{n}}$
$n \geq \left(\frac{z_{\alpha/2}s}{E}\right)^2$	$n \cong n_0 \left(\frac{h_0}{h}\right)^2$
$n = \left(\frac{z_{\alpha/2}}{E}\right)^2 \hat{p}(1 - \hat{p})$	$\begin{aligned} R_{i+1} &= (aR_i + c) \bmod m \\ z &= y \bmod m \\ &= y - m \left\lfloor \frac{y}{m} \right\rfloor \end{aligned}$
$\chi_0^2 = \sum_{j=1}^k \frac{(c_j - np_j)^2}{np_j}$ reject if $\chi_0^2 > \chi_{\alpha, k-s-1}^2$	$\begin{aligned} D_n^+ &= \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - \hat{F}(x_{(i)}) \right\} \\ D_n^- &= \max_{1 \leq i \leq n} \left\{ \hat{F}(x_{(i)}) - \frac{i-1}{n} \right\} \\ D_n &= \max\{D_n^+, D_n^-\} \end{aligned}$
$L = \lambda W$ $L_q = \lambda W_q$ $B = \lambda \text{E}[ST] = \frac{\lambda}{\mu}$ $L = L_q + B$ $W = W_q + \text{E}[ST]$	$P \left\{ \bigcap_{i=1}^k E_i \right\} \geq 1 - \sum_{i=1}^k \alpha_i$
$h = t_{\alpha/2, n-1} \frac{s}{\sqrt{n}}$	$\hat{p} \pm z_{\alpha/2} \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$

---

## APPENDIX G

---

### ARENA PANEL MODULES

**TABLE G 1 Basic Process Panel Modules**

---

CREATE	Used to create and introduce entities into the model according to a pattern.
DISPOSE	Used to dispose of entities once they have completed their activities within the model.
PROCESS	Used to allow an entity to experience an activity with the possible use of a resource.
ASSIGN	Used to make assignments to variables and attributes within the model.
RECORD	Used to capture and tabulate statistics within the model.
BATCH	Used to combine entities into a permanent or temporary representative entity.
SEPARATE	Used to create duplicates of an existing entity or to split a batched group of entities.
DECIDE	Used to provide alternative flow paths for an entity based on probabilistic or condition-based branching.
VARIABLE	Used to define variables for use within the model.
RESOURCE	Used to define a quantity of units of a resource that can be seized and released by entities.
QUEUE	Used to define a waiting line for entities whose flow is currently stopped within the model.
ENTITY	Used to define different entity types for use within the model.
SET	Used to define a list of elements within Arena that can be indexed by the location in the list.
SCHEDULE	Used to define a staffing schedule for resources or a time-based arrival pattern.

---

**TABLE G 2 Advanced Process Panel Modules**


---

SEIZE	Allows an entity to request a number of units of a resource. If the units are not available, the entity waits in a queue.
DELAY	Allows an entity to experience a delay in movement via the scheduling of an event.
RELEASE	Releases the units of a resource seized by an entity.
MATCH	Allows entities to wait in queues until a user specified matching criteria occurs.
HOLD	Holds entities in a queue until a signal is given or until a condition in the model is met.
SIGNAL	Signals entities in a HOLD queue to proceed.
PICKUP	Allows an entity to pick up and place other entities into a group associated with the entity.
DROPOFF	Allows an entity to drop off entities from its entity group.
SEARCH	Allows an entity to search a queue for entities that match search criteria.
REMOVE	Allows an entity to remove other entities directly from a queue.
STORE	Indicates that the entity is in a STORAGE.
UNSTORE	Indicates that the entity is no longer in a STORAGE.
ADJUST VARIABLE	Adjusts a variable to a target value at a specified rate.
READWRITE	Allows input and output to occur within the model.
FILE	Defines the characteristics of the operating system file used within a READWRITE module.
EXPRESSION	Allows the user to define named logical/mathematical expressions that can be used throughout the model.
STORAGE	Demarks a location/concept that may contain entities.
ADVANCED SET	Used to define a list of elements within Arena that can be indexed by the location in the list.
FAILURE	Used to define unscheduled capacity changes for resources according to a time pattern or a usage indicator.
STATISTIC	Used to define and manage time-based, observation-based, and replication statistics.

---

**TABLE G 3 Advanced Transfer Panel Modules**


---

STATION	Allows the marking in the model for a location to which entities can be directed for processing.
ROUTE	Facilitates the movement between stations with a time delay.
ENTER	Represents STATION, DELAY, and/or EXIT/FREE.
LEAVE	Represents ROUTE or TRANSPORT or CONVEY with DELAY option.
PICKSTATION	Allows entity to decide on its next station based on conditions.
ACCESS	Requests space on a conveyor. Entity waits if no space is available.
CONVEY	After obtaining space on a conveyor, causes the entity to be conveyed to its destination station.
EXIT	Releases space on a conveyor.
START	Causes a stopped conveyor to start transferring entities.
STOP	Causes a conveyor to stop transferring entities.
ALLOCATE	Assigns a transporter to an entity without moving the transporter. The entity now has control of the transporter. The entity may wait in queue if no transporters are available.

---

**TABLE G 3** (*Continued*)

---

MOVE	Moves an allocated transporter to a station destination.
REQUEST	Asks a transporter for a pick up. The requesting entity waits until a transporter is allocated and moves to the pickup location.
TRANSPORT	Same as REQUEST followed by MOVE to the entities desired location.
FREE	Causes the entity to release an allocated transporter.
HALT	Changes the state of the transporter to inactive.
ACTIVATE	Changes the state of the transporter to active.
SEQUENCE	Allows for prespecified routes of stations to be defined and attributes to be assigned when entities are transferred to the stations.
CONVEYOR	Defines a conveyor as a list of segments and provides the velocity and space characteristics of the conveyor.
SEGMENT	Defines the distance between two stations as a segment on a conveyor.
TRANSPORTER	Defines a mobile resource and its characteristics capable of free path or guided path movement.
DISTANCE	Defines the from-to distances between stations for free path transporters.
NETWORK	Defines a set of transporter links between intersections that represents a guided path network for transporters.
NETWORKLINK	Defines the characteristics of a space constraining path between intersections within guided path networks.
ACTIVITY AREA	Defines stations that are part of an area to facilitate the collection of aggregate statistics on the group of stations.

---

**TABLE G 4** Miscellaneous Useful Blocks, Attributes, and Variables

---

IF-ELSEIF-ELSE-ENDIF	From the Blocks panel, these modules allow standard logic-based flow of control.
WHILE-ENDWHILE	From the Blocks panel, these modules allow for iterative looping.
BRANCH	From the Blocks panel, this module allows probabilistic and condition-based path determination along with cloning of entities.
TNOW	The current simulation time.
NREP	Current replication number.
MREP	Maximum number of replications.
J	Index in SEARCH module.
NQ(queue name)	Number of entities in the named queue.
MR(resource name)	The current capacity of the named resource.
NR(resource name)	The current number of busy units of the named resource.
Entity.SerialNumber	A number assigned to an entity upon creation. Duplicates will have this same number.
Entity.Jobstep	The entity's current position in its sequence.
Entity.Sequence	The entity's sequence when using a transfer option.
Entity.Station	The entity's location or destination.
Entity.CurrentStation	The entity's location.
Entity.CreateTime	The value of TNOW when the entity was created.
IDENT	A unique number assigned to an entity while in the model. No entities have the same IDENT number.

---



# INDEX

- abline, 264
- acceptance rejection, 47
  - algorithm, 48
  - example, 49
  - probability of acceptance, 49
  - probability of rejection, 48
- accumulating conveyors, 522
- across replication statistics, 301
- active data objects, 603
- active entity, 175
- active state, 146
- activities, 113
- activity, 164
- activity cycle diagram, 420
- activity diagram, 13, 111, 444, 644
- activity-based costing, 573
- Advanced Transfer, 505
- animate
  - flow chart connectors, 131
  - queue, 130
  - resources, 129
  - toolbar, 129
  - variable, 131, 186
- Animate Transfer toolbar, 499, 520
- animation, 125
  - conveyors, 520
  - guided path transporters, 535
  - resource, 565
  - resource constrained transfer, 499
  - routes, 500
- transporters, 509
- ANOVA tests, 377
- area estimation, 72, 101
  - spreadsheet, 73
- Arena, 98
  - environment, 99
  - help, 133
  - modules, 99
  - Run Controller, 138
  - Run Setup, 102, 118
  - SMARTS files, 133
  - spreadsheet view, 99
  - toolbars, 99
- array, 170
  - 1-D, 184
  - 2D, 600
- arrayed expression, 201, 366
- arrays, 465
- arrival process, 167
- arrivals, 107
  - Poisson, 110, 115, 167
- assigning sequences, 449
- attribute, 164, 165, 172
- autocorrelation, 34, 35
  - acf, 36, 245
  - plot, 34, 255
- autocorrelation plot, 245
- automated guided vehicles, 528
- autoregressive order one, 623
- average on-hand inventory, 463

balk, 124  
 balking, 427  
 base time units, 291  
**BATCH**  
 attributes, 585  
 batch means, 346, 348  
 algorithm, 349  
 batch size, 349  
 correlated, 350  
 insufficient, 350  
 run length, 351  
 series, 349  
 time persistent, 350  
**batch/truncate**, 337, 345  
 batching entities, 210  
 Bernoulli, 238  
 beta, 241  
 bias, 325, 348  
 bias estimator, 328  
 bidirectional links, 535  
 binomial, 238  
 Blocks template, 175, 193, 203  
**Bonferroni**  
 curse, 379  
 inequality, 380  
 error bound, 380  
**bottleneck**, 444  
 box-whiskers plot, 377  
 breakpoints, 259  
 bus system, 511, 587  
 busy, 553  
 By Sequence Option, 447, 509  
 call service centers, 411  
 calling population, 395  
 cancel trace, 145  
 capacity change rule, 562  
 cash flow, 76  
 category overview, 573  
 change entity picture, 219  
 Check Model, 188  
 chi-squared test, 257, 260  
   degrees of freedom, 261  
   p-value, 261  
 chi-squared test, 26  
   2-Dimensional, 29  
 clock update phase, 146, 148  
 clone, 210  
 coefficient of variation, 236, 241, 418  
 comma separated value (csv), 338  
 common random numbers  
   CRN, 360  
   synchronization, 369  
 communicate with signals, 441  
 compare scenarios, 382  
 comparing systems, 354  
   assumptions, 355  
   output analyzer, 367  
   two, 354  
   two dependent samples, 359  
   two independent samples, 355  
 compound arrival process, 168  
 conceptual model, 13, 112, 643, 644  
 condition delayed state, 147  
 conditional delay, 113  
 confidence interval, 80  
   mean, 81  
   on difference, 355  
   proportion, 83  
 confidence level, 80  
 connect toolbar icon, 114  
 constrained transfer, 490  
 contingency table test, 245  
 continuous empirical, 277  
 conveyors, 511  
   accumulating, 511, 523  
   animation, 521  
   cell size, 514  
   cells, 513  
   constructs, 646  
   diverging, 525  
   entity size, 518  
   loading/unloading, 519  
   maximum number of cells, 518  
   merging, 525  
   modeling, 656  
   nonaccumulating, 511  
   number of cells, 514  
   processing on, 527  
   random spacing, 512  
   recirculating, 528  
   segments, 514, 517  
   simple modeling, 512  
   space, 512  
   statistics, 519, 524  
   velocity, 518  
 convolution, 46  
 correlated random variables, 623, 624  
 correlation, 34, 278  
   negative, 35  
   positive, 35, 325  
 correlogram, 36  
 costing  
   categories, 573  
   entity, 571  
   nonvalue added, 572  
   other, 572  
   resource, 568  
   transfer cost, 572  
   value added, 572  
   waiting time, 572  
 count-based failures, 562  
**COUNTER**, 594  
**COUNTIF**, 26  
 covariance, 34  
 covariance stationary, 34, 35, 326, 347

- cumulative average, 324
- cumulative average plot, 335
- cumulative distribution function, 41, 235
- current events chain, 143
- current events list, 147
- cyclical rule, 494
- database, 593
  - tables, 593
- deadlock, 534
- debug bar, 145
- debugging, 138, 145
- DECIDE, 191
  - 2-way by chance, 191
  - 2-way by condition, 191
  - N-way by chance, 191
  - N-way by condition, 191
- decision variables, 300, 427
- DEGREE, 9
- delay list, 148
- descriptive model, 6
- direct connect, 445
- direction of travel, 530
- DISC(), 104, 180
- discrete event, 106
- discrete uniform, 239
- discretize observations, 336
- distance, 502
- distribution
  - Bernoulli, 43, 64, 69
  - beta, 241, 279
  - binomial, 69, 279
  - discrete, 41
  - discrete uniform, 43
  - Erlang, 47
  - exponential, 39
  - gamma, 260
  - geometric, 43, 64
  - hyper-exponential, 51
  - lognormal, 241
  - mixture, 50
  - normal, 66, 241
  - others, 292
  - Poisson, 43
  - shifted, 53
  - shifted geometric, 43
  - triangular, 240
  - truncated, 52
  - uniform, 40, 64
  - Weibull, 287
- distribution fitting
  - continuous, 254
  - discrete, 244
- distributions, 100
- dormant, 147
- down time, 564
- drawing editor, 127
- drawing toolbar, 499
- drawing tools, 127
- drop off logic, 588
- DSTAT, 314, 594
- duplicate, 210
- duplicate original, 215
- effective arrival rate, 406, 413
- empirical distribution, 31, 263
  - continuity correction, 31
- empty and idle, 327
- end of file action, 599
- ENTER
  - free transporter, 507
  - transfer in option, 496
  - transporters, 507
- entity, 111, 164
  - active, 146, 183
  - cost modeling, 573
  - costing, 571
  - duplicate, 431
  - group, 585
  - group functions, 586
  - holding cost, 573
  - initial cost, 572
  - logical, 166, 281, 430
  - permanent, 210, 585
  - picture, 125
  - representative, 210, 216
  - states, 146
  - temporary, 585
  - transfer, 490
- entity movement phase, 148
- entity transfer, 445, 537
- Entity.CurrentStation, 447
- Entity.JobStep, 447
- Entity.Picture, 173
- Entity.PlannedStation, 447
- Entity.Sequence, 447, 449
- Entity.SerialNumber, 172, 190, 433
- Entity.Station, 447
- Entity.Type, 173
- Erlang delay probability, 410
- Erlang loss formula, 414
- error tolerance, 382
- escalator, 512
- event calendar, 146
- events, 109, 113
  - calendar, 143
  - future, 109
  - schedule, 109
  - scheduling, 137
- expected number in queue, 398
- expected total cost per time, 416
- expected value, 235
- expected waiting time, 398
- experimental design, 10, 665
- experimental factors, 373
- exponential smoothing, 335

expression builder, 180  
 external files, 597  
 F-test for equal variances, 355  
 factor blocking, 361  
 failed, 553  
 failures  
     multiple, 564  
 fast forward, 119  
 FCFS, 396  
 files  
     Excel, 601  
     text, 599  
 fill rate, 457  
 finite horizon, 306  
 finite population, 414  
 finite queue, 412  
 first-in first out, 305  
 fitdist, 251  
 fitdistrplus, 250  
 floor function, 20  
 flow chart, 101  
 flow of control, 190  
 free path transporter, 501  
 frequency  
     empty category, 566  
     OUTPUT, 568  
     statistics, 566  
 frequency tabulation, 566  
 future events heap, 143  
 future events list, 148, 164  
 general distribution, 396  
 generic station, 579  
 geometric, 239  
 gofstat, 251  
 goodness of fit, 257, 265  
 GOTO iterative loop, 209  
 guided path network, 530  
 guided path transporter, 501  
 half-width, 81, 121, 309  
 half-width error bound, 312  
 half-width ratio method, 309  
 histogram, 245  
     intervals, 257  
 HOLD  
     infinite, 587  
     infinite hold, 656  
     release limit, 438  
 HOLD/SIGNAL  
     communication, 439  
     example, 437  
 holding cost, 457  
 IDENT attribute, 172, 190  
 idle, 553  
 if blocks, 194  
 ignore, 556  
 inactive, 553  
 independence, 255  
 independent sampling, 369  
 indicator variable, 83, 291, 308  
 indifference zone, 358  
 infinite hold, 148, 435, 436  
 infinite horizon, 306, 321  
 infinite loop, 209  
 initial conditions, 300, 327  
 initialization bias, 331, 661  
 initialization bias problem, 328  
 Input Analyzer, 267  
     fit all, 269, 272  
 input distribution modeling, 242  
 input modeling process, 243  
 intersection, 529  
 inventory  
     (r,Q) policy, 453  
     back orders, 454  
     level, 455  
     on order, 455  
     performance measures, 455  
     position, 455  
     systems, 394, 453  
 inventory model, 73  
 inverse transform, 38  
     algorithm, 39  
     Bernoulli, 43  
     discrete, 41  
     discrete uniform, 43  
     exponential, 39  
     geometric, 43  
     theorem, 38  
     uniform, 41  
 iteration, 9  
 iterative looping, 195  
 J variable, 434  
 job step, 447, 451  
 joint mass function, 249  
 K-S Test, 262  
 K-S Test, 30  
     statistic, 32  
 Kendall notation, 396  
 Kolmogorov–Smirnov (K-S) Test, 30  
 kurtosis, 236  
 lag-k autocorrelation, 326  
 lead-time, 455  
 LEAVE  
     connect type, 496  
     transfer out, 495  
     transporters, 506  
 Lindley's equation, 322  
 linear congruential generator (LCG), 19  
     increment, 19  
     modulus, 19

multiplier, 19  
properties, 20  
seed, 19  
theorem, 21  
link, 529  
  bidirectional, 529  
  unidirectional, 529  
Little's formula, 398, 400, 405  
loading, 505  
log-likelihood function, 249  
logical branching, 191  
logical entity, 462, 545  
logical operators, 191  
lognormal, 241  
LOOKUP(), 78  
LOTR Makers, 279  
  
M/M/1, 121, 396, 437  
  example, 407  
M/M/c, 406  
  example, 410  
M/M/c/c, 414  
M/M/c/k, 413  
M/M/c/k/k, 415  
machine interference, 415  
  model, 419  
machine utilization, 417  
majorizing function, 47, 50  
MakeWelchData, 342  
Markovian, 408  
Markovian assumptions, 401  
material-handling, 512  
math operators, 191  
max arrivals, 187  
maximum likelihood method, 249  
mean arrival rate, 548  
mean number turned away, 413  
mean system capacity, 401  
mean time between failures, 638  
mean time to repair, 638  
mean value theorem, 303  
median, 237  
MEMBER function, 364  
method of batch means, 346  
method of independent replications, 307  
method of moments, 249, 260  
method of replication deletion, 331  
mixed congruential generator, 24  
mixture distribution, 50  
  generating from, 50  
mod operator, 20  
model initialization options, 660  
model translation, 13  
modeling issues, 646  
modeling recipe, 177, 198, 211, 641  
Modules  
  ACCESS, 513  
  ACTIVATE, 502  
ALLOCATE, 502  
ALTER, 552  
ASSIGN, 100, 180  
BATCH, 210, 216, 585  
CONVEY, 513  
CONVEYOR, 513  
CREATE, 100, 114, 137, 166  
DECIDE, 123  
DELAY, 118, 205  
DISPOSE, 100, 168  
DISTANCE, 503  
DROPOFF, 585  
ENDWHILE, 199  
ENTER, 493  
ENTITY, 115, 126, 573  
EXIT, 513  
EXPRESSION, 196, 200  
FILE, 182, 603  
FREE, 502  
HALT, 502  
HOLD, 435  
INTERSECTION, 530  
LEAVE, 495  
LINK, 530  
MOVE, 502  
NETWORK, 530  
OUTPUT statistic, 424  
PICKSTATION, 576  
PICKUP, 585  
PROCESS, 117  
READWRITE, 177, 182, 597  
RECORD, 100, 184, 312  
RELEASE, 113, 118  
REMOVE, 431  
REQUEST, 502  
RESOURCE, 116  
ROUTE, 442, 446  
SCHEDULE, 550  
SEARCH, 431, 587  
SEGMENT, 514  
SEIZE, 113, 118  
SEPARATE, 210, 211, 431  
SEQUENCE, 442, 446  
SET, 220  
SIGNAL, 435  
START, 513  
STATION, 442, 445  
STATISTIC, 313  
STOP, 513  
TRANSPORT, 502, 508  
TRANSPORTER, 501, 502  
VARIABLE, 100, 170  
WHILE, 199  
Monte Carlo, 71, 100  
  integration, 71  
  method, 71  
moving average, 335, 337  
MR() function, 421

- MREP, 319
- multi-echelon inventory system, 464
- multiple comparisons, 372
- multiple comparisons with the best (MCB), 379
- named range, 601
- naming convention, 170, 174, 223
- negative binomial, 239
- negative correlation
  - effects, 348
- network, 530
- networks of queues, 442
- news vendor model, 73, 104
  - pseudo-code, 105
- NG number in group, 587
- non-Markovian, 418
- non-stationary, 278, 544
  - arrivals, 544
- non-stationary arrivals, 655
- non-stationary, 306, 325
- NORM.INV(), 66
- normal, 241
- normal-to-anything transformation (NORTA), 623
- NQ() function, 123
- NREP, 319
- objective function, 426
- offered load, 401, 653
- optimization model, 426
- OptQuest, 419, 424, 664, 670
- order frequency, 457
- order statistics, 237
- ordering cost, 457
- ORUNHALF, 319
- OUTPUT, 594
- output analysis, 305
- output analyzer, 335
- OUTPUT statistic, 290
- overall decision error, 381
- P-P plot, 263
- p-value
  - chi-squared test, 27
- p-value, 25
  - interpretation, 25
- pair-t confidence interval, 359
- pairwise comparisons, 381
- PAN file, 374
- parallel processing, 213
- parameters, 164
- pharmacy model, 110, 176
- pick up logic, 588
- PICKSTATION
  - active stations, 579
  - criteria, 577
- PICKUP/DROPOFF, 656
- picture library, 130, 499
- piece-wise constant-rate function, 549
- plotdist, 250
- PMMLCG, 21
- point estimate, 80
- Poisson, 239
  - exponential relationship, 45
  - fitting, 244
  - splitting, 467
- Poisson Arrivals See Time Averages, 460
- Poisson demand, 467
- Poisson process fitting, 546
- pooled variance, 357
- positive correlation
  - effects, 348
- practical significance, 358
- preempt, 556
- preempt rule, 559
- prescriptive model, 6
- present value, 77
- prime modulus multiplicative linear congruential generator, 21
- priority, 366
- priority processing, 430
- probability density function, 234
- probability distribution, 234
- probability mass function, 41, 234
- probability of overtime, 280
- problem formulation, 9, 11
- process, 111
- Process Analyzer, 372, 374, 663
  - Chart Wizard, 377
  - run scenarios, 376
  - scenario, 374
- process description, 644
- process-oriented view, 146
- program file (p), 374
- project report outline, 672
- pseudo random numbers
  - algorithms, 19
  - LCG, 19
  - sequence, 20
  - stream, 22
- pseudo-code, 178
  - machine interference model, 420
  - SM Testing, 649
  - test and repair, 444
- pseudorandom numbers, 18
  - generation, 19
  - sequence, 22
- Q-Q plot, 263
- QTSPPlus, 407, 413, 416
- quantile, 263
- quantile function, 264
- quartile, 238
  - first, 238
  - third, 238
- query, 595
- QUEUE

- ranking, 432
- queue, 164
  - network, 442
  - notation, 396
  - performance measures, 396
  - ranked, 432
  - single line, 394
- queue discipline, 305, 395
- queue statistics, 305
- queuing
  - analysis software, 407
  - approximations, 418
  - systems, 394
- RAND(), 64
- RANDBETWEEN(), 64
- random
  - numbers, 18
  - pseudo, 18
  - variable, 18, 37
  - variate, 18, 37
- random number stream, 369
- random numbers
  - independence, 34
  - pattern testing, 37
  - runs tests, 37
  - testing, 24
- random sample, 307
- random variable, 233
  - range, 233
- randomness, 7
- ranked queue, 432
- ranking and selection, 381
- rate inversion, 546
- reading from Access, 605
- ready state, 146
- recirculation, 528
- RECORD module
  - data capture, 333
- RECORD module options, 314
- Recordsets, 603
- relative comparison, 452
- relative frequency, 258
- reneging, 427, 430
- reorder point, 455
- reorder quantity, 455
- reordering logic, 459
- replenishment, 455
- replenishment logic, 461
- replication, 300
- replication length, 188, 335
- replication–deletion, 332
- report viewer, 120
- reports
  - database, 593
- representative entity, 216, 585
- resource, 111, 164
  - active view, 440
  - as entity, 440
  - capacity, 553
  - cost calculation, 570
  - costing, 568
  - definition, 643
  - failures, 562
  - inactive, 579
  - mobile, 529
  - MR(), 552
  - NR(), 553
  - sets, 362, 364
  - sharing, 362
  - states, 552, 565
- resource selection rule, 494
- resource selection rules, 364
- resource set, 491
  - defining, 366
- resource states, 129
- resource statistics, 561
- resource-constrained transfer, 491
- retailer, 467
- ride point, 511
- ring diameter, 287
- Run Controller Commands, 139
- run length, 334
  - setting, 350
- Run Setup
  - statistical collection, 315
- run toolbar, 118
- sample
  - average, 35, 236
  - covariance, 35
  - error, 79, 104
  - size, 81
  - space, 233
  - variance, 35, 81, 236
- sample path, 300
- sample size determination, 309
- sampling error, 73
- save attribute, 366, 508, 582
- scan for condition, 435, 436
- scatter plot matrix, 246
- SCHEDULE
  - arrival rate per hour, 550
- schedule
  - infinite duration, 555
- schedule rule, 556
- scheduled capacity changes, 554
- schedules
  - defining, 555
- SEIZE
  - priority, 365, 496
  - select queue, 583
- seize area, 130
- seize priority, 495
- selecting the best system, 668
- sensitivity analysis, 278, 372, 376

sequence dependent, 443  
 sequential sampling, 318  
     batch means, 350, 352  
 server, 396  
     utilization, 401  
 service, 108  
 SET WATCH, 143  
 sets  
     functions, 364  
     modeling, 580  
     queue, 582  
     sequence, 448  
     station, 581  
     tally, 591  
 shifted distribution, 53, 270  
     example, 53  
 shortest path matrix, 536  
 SIGNAL  
     release limit, 438  
 SIMAN, 98, 134  
     exp file, 134  
     mod file, 134  
 SIMAN object, 616  
 simulation, 1  
     advantages, 3, 17  
     black box, 299  
     clock, 110  
     definition, 2  
     discrete event, 106  
     input, 299  
     languages, 7  
     methodology, 8  
     output, 299  
     run length, 290  
     spreadsheet, 63  
 skewness, 236, 257  
 SMART files, 523  
 split existing batch, 215  
 spreadsheet  
     PV() function, 78  
 spreadsheets  
     challenges, 70  
 spur, 529, 535  
 square root rule, 258  
 square root staffing rule, 411  
 squared error criterion, 271  
 staging area, 511  
 state  
     active, 146  
     condition delayed, 147  
     dormant, 147  
     ready, 146  
     time delayed, 146  
 state-transition diagram, 404, 415  
 STATE() function, 553  
 STATESET, 501  
 station  
     beginning, 517  
     generic, 579  
     index, 582  
     markers, 500  
     set members, 582  
 statistical variables, 300  
 statistically significant vs practical significance, 358  
 statistics  
     across replication, 595  
     FREQUENCY, 595  
     OUTPUT, 595  
     steady state, 306, 327, 330, 403  
         cyclical, 307  
         distribution, 327  
         equations, 405  
         probabilities, 405  
         probability, 403  
     steady-state cyclical parameter estimation, 660  
     stock out, 457  
     stock out indicator, 457  
     stopping condition, 306  
     stream control, 371  
     stream parameter, 200  
     Student-t, 81  
     Sturges rule, 258  
     sub-models, 196  
     submodel, 203, 433  
         process based, 206  
     supply chain, 464  
     SymbolNumber, 616  
     symmetric, 503  
     system, 3, 110, 113, 163  
         conceptual, 4  
         conceptualization, 4  
         continuous, 4  
         definition, 3, 12  
         deterministic, 4  
         discrete, 4  
         dynamic, 4  
         physical, 4  
         state, 109, 164  
         static, 4  
         stochastic, 4  
         types, 4  
     t-distribution, 81  
     T.INV(), 81  
     TALLY, 302, 594  
     tally-based, 307  
         Arena collection, 314  
     TAVG, 319  
     terminating, 306  
     test for time dependence, 247  
     test plan, 451  
     testing independent configurations, 357  
     text reports, 121  
     TFIN, 607  
     THALF, 352  
     thinning, 546

- throughput, 358
- rate, 399
- Tie-Dye T-Shirts, 211
- time horizon, 659
- time series plot, 245
- time units, 205
- time varying, 544
- time-based failures, 562
- time-delayed state, 146
- time-average, 303, 337
  - weighted, 304
- time-persistent, 302, 307
  - Arena collection, 312
  - data capture, 334
  - filtering, 336
- TNOW, 181, 183, 290
- tracing, 139
- trade-off, 14
- transient probability, 403
- transporter
  - free path, 501
  - guided path, 501, 529
  - initial position, 505
- transporter selection rules, 508
- transporters, 501
  - variables, 536
- triangular, 240
- truncated distribution, 52
  - algorithm, 52
  - example, 53
- type 1 error, 25, 81
- unbiased estimator, 328
- unconstrained transfer, 490
- uniform, 240
- unit cost, 457
- unloading, 505
- up time, 564
- utilization, 124, 401
  - calculating, 561
  - instantaneous, 561
  - schedule, 561
- validation, 9, 14
- value-added cost, 573
- value-added processing, 574
- variable
  - array, 178
- variables, 164, 165
  - global, 170
  - initial value, 171
- variance, 235
  - estimator, 361
  - pooled estimator, 356
- variance reduction, 361, 369
- VBA, 609
  - editor, 610
  - module, 610
- VBA event model, 609
- VBA event routines, 609
- VBA message box, 612
- VBA user form, 618
- VBA user-defined function, 614, 619
- velocity, 502
- velocity change factor, 529
- verification, 9, 13, 652
- VIEW QUEUE, 143
- Visual Basic, 593
- visual basic, 69
- wait, 556
- wait for signal, 435, 436
- wait rule, 558
- waiting time in queue, 303
- walk-in clinic, 429
- walking speed, 504
- warehouse, 467
- warm-up period, 328, 329
  - setting, 345
- Welch plot, 330, 334, 661
  - macro, 340
  - Microsoft Access, 342
  - spreadsheet, 341
- while loop, 203
- Winter Simulation Conference, 675
- within replication
  - observations, 327
  - statistics, 300
- work conserving, 397
- work sampling, 544
- writing output as text, 338
- zero transfer delay, 459
- zone, 529
  - control, 531, 533
  - control rule, 533

# **WILEY END USER LICENSE AGREEMENT**

Go to [www.wiley.com/go/eula](http://www.wiley.com/go/eula) to access Wiley's ebook EULA.