

Simulation Modeling and Arena

Manuel D. Rossetti

2021-06-09

Contents

1. Simulation Modeling	1
1.1. Simulation Modeling	1
1.2. Why Simulate?	2
1.3. Types of Systems and Simulation Models	4
1.4. Simulation: Descriptive or Prescriptive Modeling?	8
1.5. Randomness in Simulation	9
1.6. Simulation Languages	9
1.7. Simulation Methodology	11
1.8. Organization of the Book	18
1.9. Exercises	19
2. Introduction to Simulation and Arena	21
2.1. The Arena Environment	21
2.2. Performing Simple Monte-Carlo Simulations using Arena	22
2.2.1. Simple Monte Carlo Integration {ssMC}	24
2.2.2. Arena Modules Needed for Static Simulation Examples	26
2.2.3. Area Estimation via Arena	27
2.2.4. The News Vendor Problem	31
2.3. How the Discrete-Event Clock Works	36
2.4. Simulating a Queueing System By Hand	40
2.5. Elements of Process-Oriented Simulation	49
2.5.1. Entities, Attributes, and Variables	50
2.5.2. Creating and Disposing of Entities	52
2.5.3. Defining Variables and Attributes	55
2.6. Modeling a Simple Discrete-Event Dynamic System	61
2.6.1. A Drive through Pharmacy	61
2.6.2. Modeling the System	61
2.6.3. Pharmacy Model Implementation	65
2.6.4. Specify the Arrival Process	66
2.6.5. Specifying the Resources	67
2.6.6. Specify the Process	68
2.6.7. Specify Run Parameters	70
2.6.8. Analyze the Results	72
2.7. Extending the Drive Through Pharmacy Model	76
2.8. Animating the Drive Through Pharmacy Model	78

Contents

2.9.	Attributes, Variables, and Some I/O	87
2.9.1.	Modifying the Pharmacy Model	87
2.9.2.	Using the ASSIGN Module	91
2.9.3.	Using the READWRITE Module	94
2.9.4.	Using the RECORD Module	97
2.9.5.	Animating a Variable	98
2.9.6.	Running the Model	100
2.10.	How Arena Manages Entities and Events	104
2.11.	Summary	108
2.12.	Exercises	110
3.	Statistical Analysis for Finite Horizon Simulation Models	125
3.1.	Finite versus Infinite Horizon Simulation Studies	126
3.2.	Types of Statistical Quantities in Simulation	128
3.2.1.	Within Replication Observations	129
3.2.2.	Across Replication Observations	135
3.3.	Review of Statistical Concepts	137
3.3.1.	Point Estimates and Confidence Intervals	138
3.3.2.	Sample Size Determination	140
3.3.3.	Determining the Sample Size for an Arena Simulation	142
3.4.	Modeling a STEM Career Mixer	146
3.4.1.	Conceptualizing the System	148
3.4.2.	Implementing the Model	160
3.4.3.	Planning the Sample Size	168
3.5.	Using Sequential Sampling Methods on a Finite Horizon Simulation	173
3.6.	Tabulating Frequencies using the STATISTIC Module	177
3.7.	Summary	182
3.8.	Exercises	185
4.	Modeling Systems with Processes and Basic Entity Flow	191
4.1.	Enhancing the STEM Career Mixer Example	192
4.1.1.	Turning Off a CREATE Module	194
4.1.2.	Modeling Walking Time	194
4.1.3.	Using Expressions within an Arena Model	196
4.1.4.	Introducing the STATION, ROUTE, STORAGE, and SET Modules	196
4.1.5.	Controlling Randomness by Specifying Stream Numbers	199
4.1.6.	Pseudo-code for the Revised STEM Mixer Example	201
4.1.7.	Implementing the Revised STEM Mixer Model in Arena	207
4.2.	Example: Iterative Looping, Expressions, and Sub-models	215
4.3.	Batching and Separating Entities	230
4.3.1.	Conceptualizing the Model	231
4.3.2.	Building the Model	234
4.4.	Statistical Issues When Comparing Two Systems	242
4.4.1.	Analyzing Two Independent Samples	244

4.4.2. Analyzing Two Dependent Samples	248
4.5. The LOTR Makers, Inc. Example	251
4.5.1. Conceptualizing the Model	252
4.5.2. Implementing the Model	255
4.5.3. Running the Model	263
4.6. Comparing Two Alternative Configurations for the LOTR Makers	265
4.6.1. Resource Sets	267
4.7. Modeling Systems with Routing Sequences	277
4.7.1. Computer Test and Repair Shop Example	278
4.7.2. Conceptualizing the Model	279
4.7.3. STATION, ROUTE, and SEQUENCE Modules	281
4.7.4. Running the Test and Repair Model	286
4.8. Summary	289
4.9. Exercises	292
5. Statistical Analysis for Infinite Horizon Simulation Models	307
5.1. A Spreadsheet Example	308
5.2. Statistical Analysis Techniques for Warmup Detection	314
5.2.1. Assessing the Effect of Initial Conditions	314
5.2.2. Using a Welch Plot to Detect the Warmup Period	317
5.3. Performing the Method of Replication-Deletion	320
5.3.1. Looking for the Warm up Period in the Welch Plot Analyzer	325
5.4. The Batch Means Method	332
5.4.1. Performing the Method of Batch Means	336
5.5. Applying Queueing Theory Results to Verify and Validate a Simulation	340
5.5.1. Analyzing the Preparation Station	342
5.5.2. Analyzing the Build Lines	343
5.5.3. Analyzing the Packaging Station	344
5.5.4. Analyzing the Palletizing Station	345
5.5.5. Analyzing the Total System Time	345
5.5.6. Other Issues for Verification and Validation	347
5.6. Summary	349
5.7. Exercises	351
6. Modeling Systems with Advanced Process Concepts	361
6.1. Non-stationary Processes	362
6.1.1. Thinning Method	366
6.1.2. Rate Inversion Method	367
6.2. Advanced Resource Modeling	369
6.2.1. Scheduled Capacity Changes	371
6.2.2. Calculating Utilization	376
6.2.3. Resource Failure Modeling	380
6.3. Job Fair Example with Non-Stationary Arrivals	383
6.3.1. Collecting Statistics by Time Periods	385

Contents

6.3.2. Modeling the Statistical Collection	389
6.3.3. Implementing the Model in Arena	391
6.4. Modeling Balking and Reneging	399
6.5. Holding and Signaling Entities	407
6.5.1. Redoing the M/M/1 Model with HOLD/SIGNAL	408
6.5.2. Using Wait and Signal to Release Entities	414
6.5.3. Modeling a Reorder Point, Reorder Quantity Inventory Policy	418
6.6. Miscellaneous Modeling Concepts	429
6.6.1. Picking Between Stations	429
6.6.2. Generic Station Modeling	433
6.6.3. Picking up and Dropping Off Entities	438
6.7. Summary	446
6.8. Exercises	448
7. Modeling Systems with Entity Movement and Material Handling Constructs	459
7.1. Resource Constrained Transfer	460
7.1.1. Implementing Resource Constrained Transfer	463
7.1.2. Animating Resource Constrained Transfer	470
7.2. Constrained Transfer with Transporters	473
7.2.1. Test and Repair Shop with Workers as Transporters	476
7.2.2. Animating Transporters	482
7.3. Modeling Systems with Conveyors	485
7.3.1. Test and Repair Shop with Conveyors	490
7.3.2. Animating Conveyors	495
7.3.3. Miscellaneous Issues in Conveyor Modeling	497
7.4. Modeling Guided Path Transporters	505
7.5. Summary	514
7.6. Exercises	515
8. Applications of Simulation Modeling	523
8.1. Analyzing Multiple Systems	524
8.1.1. Sensitivity Analysis Using the Process Analyzer	525
8.1.2. Multiple Comparisons with the Best	532
8.2. SM Testing Contest Problem Description	538
8.3. Answering the Basic Modeling Questions	545
8.4. Detailed Modeling	550
8.4.1. Conveyor and Station Modeling	551
8.4.2. Modeling Samples and the Test Cells	553
8.4.3. Modeling Sample Holders and the Load/Unload Area	561
8.4.4. Performance Measure Modeling	564
8.5. Simulation Horizon and Run Parameters	566
8.6. Preliminary Experimental Analysis	569
8.7. Final Experimental Analysis and Results	571
8.7.1. Using the Process Analyzer on the Problem	571

8.7.2. Using OptQuest on the Problem	577
8.7.3. Investigating the New Logic Alternative	577
8.7.4. Sensitivity Analysis	580
8.8. Completing the Project	580
8.9. Some Final Thoughts	583
8.10. Exercises	586
Appendix	599
A. Generating Pseudo-Random Numbers and Random Variates	599
A.1. Pseudo Random Numbers	599
A.1.1. Random Number Generators	601
A.2. Generating Random Variates from Distributions	607
A.2.1. Inverse Transform Method	608
A.2.2. Convolution	618
A.2.3. Acceptance/Rejection	621
A.2.4. Mixture Distributions, Truncated Distributions, and Shifted Random Variables	625
A.3. Summary	630
A.4. Exercises	631
B. Probability Distribution Modeling	641
B.1. Random Variables and Probability Distributions	642
B.2. Modeling with Discrete Distributions	648
B.3. Fitting Discrete Distributions	648
B.3.1. Fitting a Poisson Distribution	649
B.3.2. Visualizing the Data	650
B.3.3. Estimating the Rate Parameter for the Poisson Distribution	656
B.3.4. Chi-Squared Goodness of Fit Test for Poisson Distribution	659
B.3.5. Chi-Squared Goodness of Fit Test	660
B.3.6. Using the fitdistrplus R Package on Discrete Data	664
B.3.7. Fitting a Discrete Empirical Distribution	667
B.4. Modeling with Continuous Distributions	669
B.5. Fitting Continuous Distributions	672
B.5.1. Visualizing the Data	672
B.5.2. Statistically Summarize the Data	673
B.5.3. Hypothesizing and Testing a Distribution	676
B.5.4. Kolmogorov-Smirnov Test	683
B.5.5. Visualizing the Fit	686
B.5.6. Using the Input Analyzer	691
B.6. Testing Uniform (0,1) Pseudo-Random Numbers	703
B.6.1. Chi-Squared Goodness of Fit Tests for Pseudo-Random Numbers	703
B.6.2. Higher Dimensional Chi-Squared Test	705
B.6.3. Kolmogorov-Smirnov Test for Pseudo-Random Numbers	708

Contents

B.6.4. Testing for Independence and Patterns in Pseudo-Random Numbers	710
B.7. Additional Distribution Modeling Concepts	712
B.8. Summary	715
B.9. Exercises	716
C. Queueing Theory	721
C.1. Single Line Queueing Stations	721
C.1.1. Queueing Notation	723
C.1.2. Little's Formula	727
C.1.3. Deriving Formulas for Markovian Single Queue Systems	730
C.2. Examples and Applications of Queueing Analysis	736
C.2.1. Infinite Queue Examples	737
C.2.2. Finite Queue Examples	743
C.3. Non-Markovian Queues and Approximations	749
C.4. Summary of Queueing Formulas	751
C.4.1. M/M/1 Queue	752
C.4.2. M/M/c Queue	752
C.4.3. M/M/c/k Queue	753
C.4.4. M/G/c/c Queue	753
C.4.5. M/M/1/k Queue	754
C.4.6. M/M/c/k Queue	754
C.4.7. M/M/1/k/k Queue	755
C.4.8. M/M/c/k/k Queue	755
C.5. Exercises	757
D. Miscellaneous Topics in Arena	763
D.1. Getting Help in Arena	763
D.2. SIMAN and the Run Controller	763
D.2.1. SIMAN MOD and EXP Files	765
D.2.2. Using the Run Controller	769
D.3. Programming Concepts within Arena	777
D.3.1. Using the Generated Access File	777
D.3.2. Working with Files, Excel, and Access	783
D.3.3. Using Visual Basic for Applications	796
D.4. Resource and Entity Costing	814
D.4.1. Resource Costing	815
D.4.2. Entity Costing	819
D.5. Summary	823
E. Arena Operators, Functions, Distributions, and Modules	825
E.1. Arena Mathematical and Logical Operators	825
E.2. Arena Probability Distributions Functions	827
E.3. Basic Process Panel Modules	828
E.4. Advanced Process Panel Modules	829

Contents

E.5. Advanced Transfer Panel Modules	830
E.6. Important SIMAN Blocks, Elements, and Pre-Defined Attributes and Variables	831
F. Distributions	833
F.1. Discrete Distrbutions	833
F.2. Continuous Distrbutions	835
G. Statistical Tables	839

Preface



This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License¹.

Welcome to the open text edition. Similarly to the previous two editions, 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 towards 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.

Release History

You are reading the 3rd edition of *Simulation Modeling and Arena* by Dr. Manuel D. Rossetti. Because of its on-line nature, updated versions of the book will be released, as needed, to correct issues and possibly add new material that would not warrant a new edition. This section summarizes noteworthy updates.

- 3rd Edition, Version 1.0, released June 14, 2021
 - first main release of text book
 - files and supporting materials are related to Arena, version 16.0

If you find typographical errors or other issues related to the text or supporting files, then please use the repository's² issue tracking system to create a new issue. You should first check if the same or similar issue has already been submitted. The issue tracking system is for filing issues about the correctness of the text or files. It is **not** about general questions about simulation concepts, solutions to homework, how to do something in Arena, etc. Such issues will not be considered and will be deleted as needed.

¹<http://creativecommons.org/licenses/by-nc-nd/4.0/>

²<https://github.com/rossetti/RossettiArenaBook>

Book Support Files

By cloning the repository or downloading the zip archive, you can have a local copy of the entire book. Thus, if you do not have regular access to internet services, you can still read and utilize the materials. I encourage students within a class setting to clone the repository using a program such as Github desktop.

The example models and related files associated with the textbook are also copyrighted under the *Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License*. You may use the model files and other supporting data and spreadsheet files at your discretion. The files come as-is, with no warranty or other obligations implied. You can find the book support files at the following links.

- Chapter Files³
- Chapter Slides⁴
- Welch Plot Application
 - Mac OS Version⁵
 - Windows Version⁶

Acknowledgments

Special thanks to the Open Educational Resources⁷ team at the University of Arkansas. This work was supported in part by a grant from the OER Course Materials Conversion Faculty Funding program at the University of Arkansas.

I would like to thank John Wiley and Sons, Inc. for their flexibility in returning to me my copyrights from previous editions of this text. This allowed this open textbook to exist. Thanks also to Rockwell Automation, Inc. for their flexibility in granting some problem materials and for cooperating with my requests to make this text open. I would like to thank colleagues that have used the previous editions of this work and their valuable feedback. I would also like to thank the students in my classes who tested version of my work and provided feedback, suggestions, and comments.

Lastly, 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 with creating figures, diagrams, and with proof-reading. Thanks so much!

³<https://uark.box.com/v/RossettiBookChapterFiles>

⁴<https://uark.box.com/v/RossettiBookChapterSlides>

⁵<https://uark.box.com/v/RossettiWelchPlotMacVersion>

⁶<https://uark.box.com/v/RossettiWelchAppWindowsVersion>

⁷<https://libraries.uark.edu/oer/>

Usage of Arena

ArenaTM is a registered trademark of Rockwell Automation, Inc and copyrighted by Rockwell Automation Technologies, Inc. This publication is not sponsored by Rockwell Automation, Inc. and Dr. Manuel Rossetti is not affiliated with Rockwell Automation, Inc. Rockwell is not responsible for providing Arena support for the textbook.

Intended Audience

This is 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 towards 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.

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 1) those that emphasize the theoretical (and mostly statistical) aspects of simulation, and 2) 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 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. After Chapter 1 the student should know what simulation is and be able to put the different types of simulation into context. Chapter 2 introduces the basics of Monte Carlo simulation and discrete event systems. It also emphasizes the important concept of how a discrete-event clock “ticks” and sets the stage for process modeling using activity diagramming. Finally, a simple (but comprehensive) example of Arena is presented so that students will feel comfortable with the tool.

Chapter 3 reviews issues in statistical concepts for the case of finite horizon simulation studies. This chapter should provide a refresher for students on statistical concepts. Chapter 4 dives deeper into process-oriented modeling. The Basic Process template within Arena is thoroughly covered. Important concepts within process-oriented modeling (e.g. entities, attributes, activities, state variables, etc.) are emphasized 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 4, student should be able to model interesting systems from a process viewpoint using Arena. Chapter 5 returns to statistical issues within the context of infinite horizon simulation. In addition, the concepts of verification and validation are discussed.

Chapters 6 and 7 present more advanced concepts within simulation and especially how Arena facilitates the modeling. In particular, non-stationary arrivals and resource staffing are introduced in Chapter 6, as well as constructs for generic station modeling, picking up and dropping off entities. Chapter 7 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.

Finally, Chapter 8 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.

Adopters of previous editions of this textbook will notice that the topics of random number generation, random variate generation, input distribution modeling, and queueing theory have been moved to the appendix. The material in those chapters is presented in a manner that is as independent as possible from the tool of Arena. This allows instructors to form a natural progression on Arena (through chapters 1-8), but also have the flexibility to cover random number generation, random variate generation and input distribution modeling if those topics are not covered in another course.

Course Syllabus Suggestion

Earlier editions of this textbook have been 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 that have never had a course in simulation may 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, etc.). I am able to cover Chapter 1-7 within a typical 16 week semester offering. A typical topic outline is as follows:

Week	Topics	Readings
1	Introduction, Generating Pseudo-Random Numbers	Chapter 1, Appendix
2	Generating Random Variates	Appendix
3	Probability Distribution Modeling	Appendix
4	Introduction to Simulation and Arena	Chapter 2
5	Introduction to Simulation and Arena	Chapter 2
6	Statistical Analysis for Finite Horizon Simulation	Chapter 3
7	Processes and Basic Entity Flow	Chapter 4
8	Processes and Basic Entity Flow	Chapter 4
9	Statistical Analysis for Infinite Horizon Simulation	Chapter 5
10	Advanced Process Concepts	Chapter 6
11	Advanced Process Concepts	Chapter 6
12	Entity Movement and Material Handling Constructs	Chapter 7
13	Entity Movement and Material Handling Constructs	Chapter 7
14	Project work	
15	Project work	
16	Project due	

I use smaller quizzes on the individual topics/chapters, team activities, homework, and a project. Note that the material is compressed to allow dedicated time for a project at the end of the semester. For instructors that do not require a project, then a less aggressive schedule can be easily achieved.

A solutions manual is available for formal adopters of the textbook. Naturally, instructors must

Contents

prove that they have adopted the book for the course. The solutions manual and associated files are copyrighted materials and should not be released to students in any form. Instructors can guess what happens when solutions become readily available to students! Contact me if you need further details.

About the Author

Dr. Manuel Rossetti⁸, P.E. Dr. Rossetti is a University Professor of Industrial Engineering⁹ and the Director of the inter-disciplinary Bachelor of Science degree in Data Science¹⁰ at the University of Arkansas. He previously served as the Director for the Center for Excellence in Logistics and Distribution¹¹ a graduated National Science Foundation Industry/University Cooperative Research Center. Dr. Rossetti has published over 120 journal and conference articles in the areas of simulation, logistics/inventory, and healthcare and has been the PI or Co-PI on funded research projects totaling over 6.4 million dollars. He was selected as a Lilly Teaching Fellow in 1997/98 and was voted Best IE Teacher by IE students in 2007, 2009, and 2017. He won the IE Department Outstanding Teacher Award in 2001-02, 2007-08, and 2010-11. He received the College of Engineering Imhoff Teaching Award in 2012 and was elected an IISE Fellow. In 2013, the UA Alumni Association awarded Dr. Rossetti the Charles and Nadine Baum Faculty Teaching Award, the highest award for teaching at the university. In 2015, Dr. Rossetti served as Program Chair for the Winter Simulation Conference¹² and will be the General Chair in 2024. He is also the author of the book, *Simulation Modeling and Arena*¹³, published by John Wiley & Sons.

Dr. Rossetti grew up in Canton, Ohio and is an ardent Cleveland sports fan. 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.

⁸<https://rossetti.uark.edu/>

⁹<https://industrial-engineering.uark.edu/>

¹⁰<https://datascience.uark.edu/>

¹¹<https://celdi.org/>

¹²<https://www.wintersim.org>

¹³<https://www.wiley.com/en-us/Simulation+Modeling+and+Arena%2C+2nd+Edition-p-9781118607916>

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

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.

1.1. Simulation Modeling

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.

1. Simulation Modeling

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) define 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 (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 that are waiting.
- The average waiting time of the patients and their average total time in the emergency department.
- The average number 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 to 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 queueing models. Later chapters of this book will present more about queueing models; however, for the present situation, the application of queueing 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 queueing models. Queueing 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.

Simulation may be the preferred modeling methodology if the understanding gained from developing and using a simulation model is worth the time and cost associated with developing and using the model. Good uses of simulation include:

- Understanding how complex interactions in the system effect performance.
- Understanding how randomness effects performance.
- Comparing a fixed set of design alternatives to determine which design meets the performance goals under which conditions
- Training people to prepare them for dealing with events that may be disruptive to the actual system.
- The model will be used repeatedly for decision making.
- When the decision associated with the problem has a high cost so that the cost of building the model and evaluating the design is worth its development.
- When the current system does not yet exist and you need to ensure that the chosen design will meet specifications.

Simulation modeling activities encapsulate all three major modeling methods of data analytics: descriptive, predictive, and prescriptive. Descriptive modeling uses historical data to describe what happened in order to understand past behavior of a system. Predictive modeling uses historical data to develop models that help us understand future behavior in order to answer what

1. *Simulation Modeling*

may happen. Descriptive modeling summarizes past data for understanding. Predictive modeling uses past data to predict future behavior. Prescriptive modeling indicates what should be done and is integral to answering questions involving system design.

A simulation model is both a descriptive and predictive model. In addition, when coupled with stochastic optimization methods or a rigorous design process that evaluates and recommends designs, a simulation model becomes an integral part of the prescriptive modeling process. A simulation model *describes* how a system works by encapsulating that description within the operating runs/constructs of the model. A simulation model uses descriptive models (input models, summary statistics). A simulation model *predicts* future system response. A simulation model can be used to predict future behavior through running what-if scenarios. Simulation is inherently a predictive modeling methodology. Unlike other predictive modeling techniques found in data analytics, such as regression, neural networks, random forests, etc., simulation explicitly incorporates *domain* knowledge into the modeling activity by incorporating system operating behavior. The behavior is modeled through the physical and logical rules that apply to the relationships between the components of the system. Unlike pure statistical predictive models, simulation has the advantage of explicitly representing relationships rather than just relying on discovering relationships. A key advantage of simulation modeling is that it has the capability of modeling the entire system and its complex inter-relationships. 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 Systems and Simulation Models

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 inter-related components working together towards 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." (Command, 1991)

1.3. Types of Systems and Simulation Models

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 upon 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.

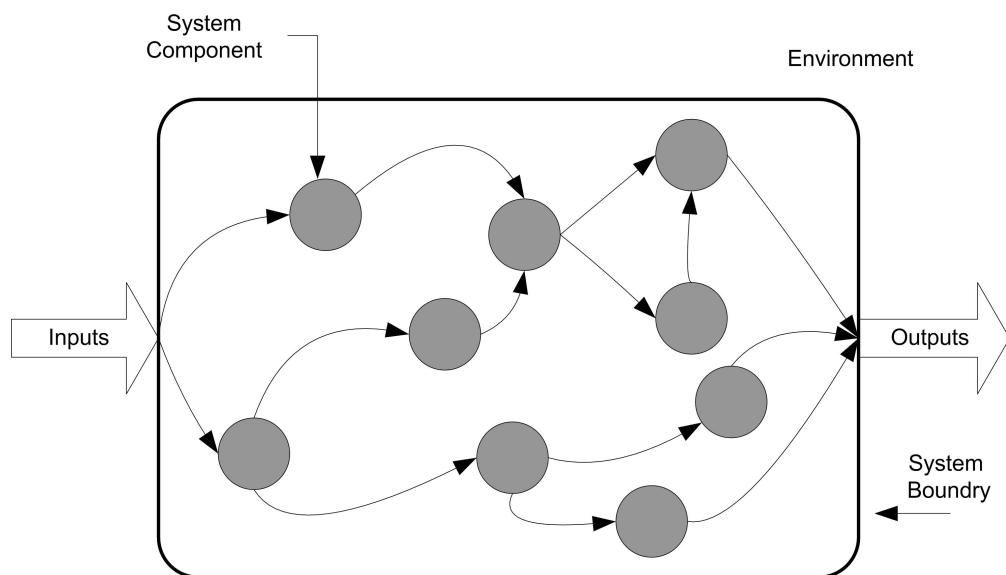


Figure 1.1.: Conceptualization of a system

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 a system as discrete, serves our purposes then you can

1. Simulation Modeling

call the system discrete. Figure 1.2 illustrates this classification of systems. This book primarily examines stochastic, dynamic, discrete systems.

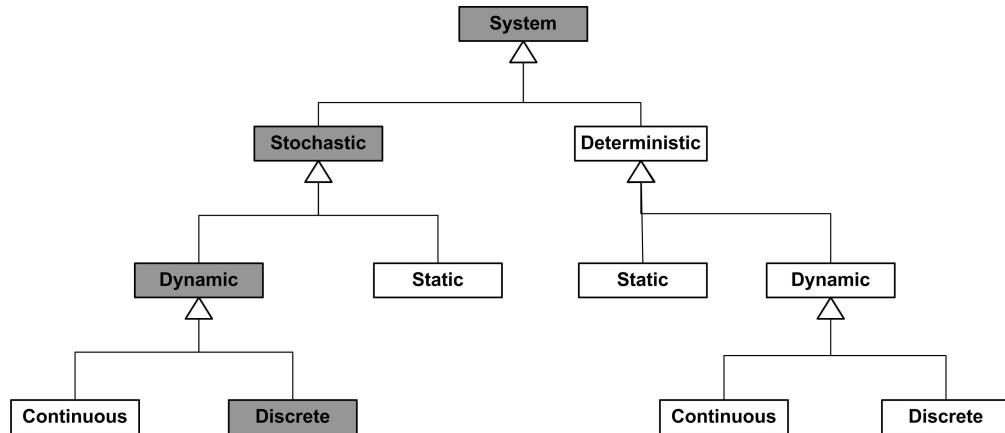


Figure 1.2.: General types of systems

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: 1) discrete event and 2) continuous.

Just as discrete systems change at discrete points in time, in 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, 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 when a customer either 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

1.3. Types of Systems and Simulation Models

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.

A system and our resulting model depends on how we characterize the state of the system. The **state** of a system is the set of properties/variables that describe the system at any time $\{x_1(t), x_2(t), \dots\}$ where $x_1(t)$ is a variable that represents a system property value at time t . Variables that take on a countable set of values are **discrete**. Variables that take on an uncountable set of values are said to be **continuous**. For discrete variables, we can define a mapping from the set of integers to each possible value. Even in the discrete case, there may be an infinite number of values. Continuous variables are represented by the set of real numbers. That is, there are an unaccountably infinite number of possible values that the variable can take on. A discrete system has all discrete variables in its state. A continuous system has all continuous variables in its state. A combined continuous-discrete system has both types of variables in defining its state. Consider an airplane:

- If we are interested in the number of parts operating to specification at any time t , then the state is $\{N_1(t), N_2(t), \dots\}$ where $N_1(t) = 1$ if part 1 is operating and 0 if part 1 is not operating to specification. The state vector consists of all discrete variables. This is a discrete system.
- If we are interested in the temperature of each part at time t , then the state is $\{T_1(t), T_2(t), \dots\}$, where $T_1(t)$ is the temperature in Celsius of part 1 at time t , etc. The state vector consists of all continuous variables. This is continuous system.
- If we are interested in the velocity of a plane at time t and the number of wheels deployed at time t , then the state is $\{v(t), n(t)\}$ where $v(t)$ is the velocity of the plane in meters/second and $n(t)$ is $\{0, 1, 2, 3, 4\}$ wheels. Then the state vector consists of both continuous and discrete variables that change with time. This is a combined continuous/discrete system.

Static systems are systems for which time is not a significant factor. In other words, that the state does not evolve over time. Dynamic systems are systems for which system state changes with respect to time. In a deterministic system, the variables are not governed by underlying random processes. In a stochastic system, some of the variables are governed by underlying random processes. Time can change continuously or at discrete points. When time changes only at discrete points in time, we call these points, events.

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. We will be modeling discrete-event dynamic stochastic systems in this textbook.

1. Simulation 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.

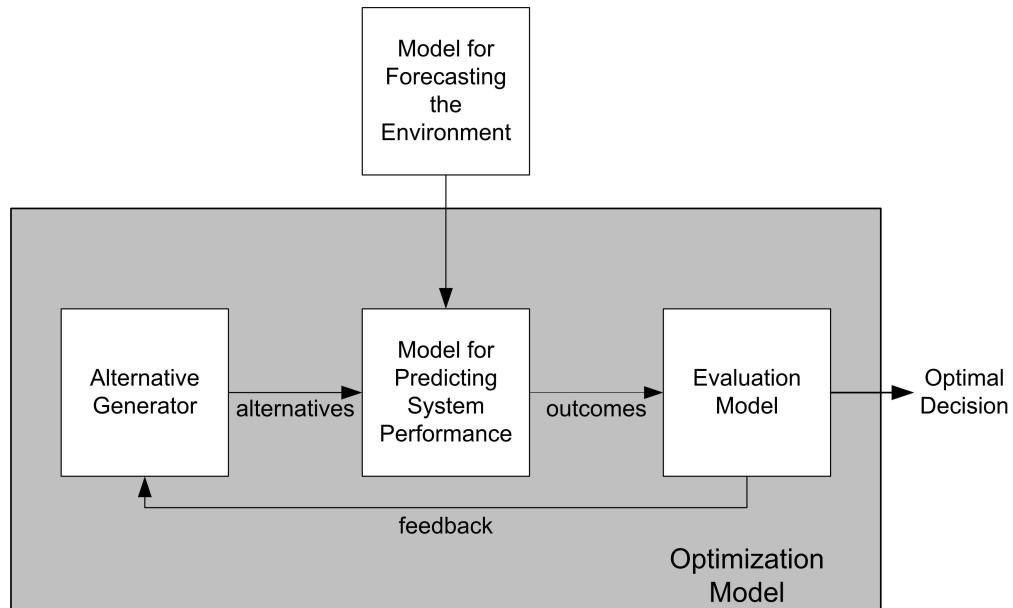


Figure 1.3.: Using simulation for prescriptive analysis

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 feed 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 the 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 is met. In this simple situation, you might act as your own alternative generator and the evaluative model is as 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 fashion. 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 inter-arrival 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 re-drive 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. Appendix B discusses 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 inter-arrival 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 is presented in Appendix A.

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

1. Simulation Modeling

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 library¹ for simulating discrete-event systems in Java is available from the author, see (Rossetti, 2008) and the related book².

The computational power and storage capacity of computers has 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) and others are more "drag and drop" (e.g. ProModel, Arena, etc.).

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). 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>).

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. 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.

is fundamentally a process description based language. That is, when using , the modeler describes the process that an "entity" experiences while flowing through or using the elements of the system. You will learn about how facilitates process modeling throughout this textbook.

¹<https://git.uark.edu/jslfork/JSL>

²<https://rossetti.git-pages.uark.edu/jslbookdownbook/>

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, i.e. 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 over looked. 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.

1. Simulation Modeling

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

1. Problem Formulation

1. Define the problem
2. Define the system
3. Establish performance metrics
4. Build conceptual model
5. Document model assumptions

2. Simulation Model Building

1. Model translation
2. Input data modeling
3. Verification
4. Validation

3. Experimental Design and Analysis

1. Preliminary Runs
2. Final experiments
3. Analysis of results

4. Evaluate and Iterate

1. Documentation
2. Model manual
3. 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,

1.7. Simulation Methodology

and implementation recognizes that simulation projects often fail if there is no follow through on the recommended solutions.

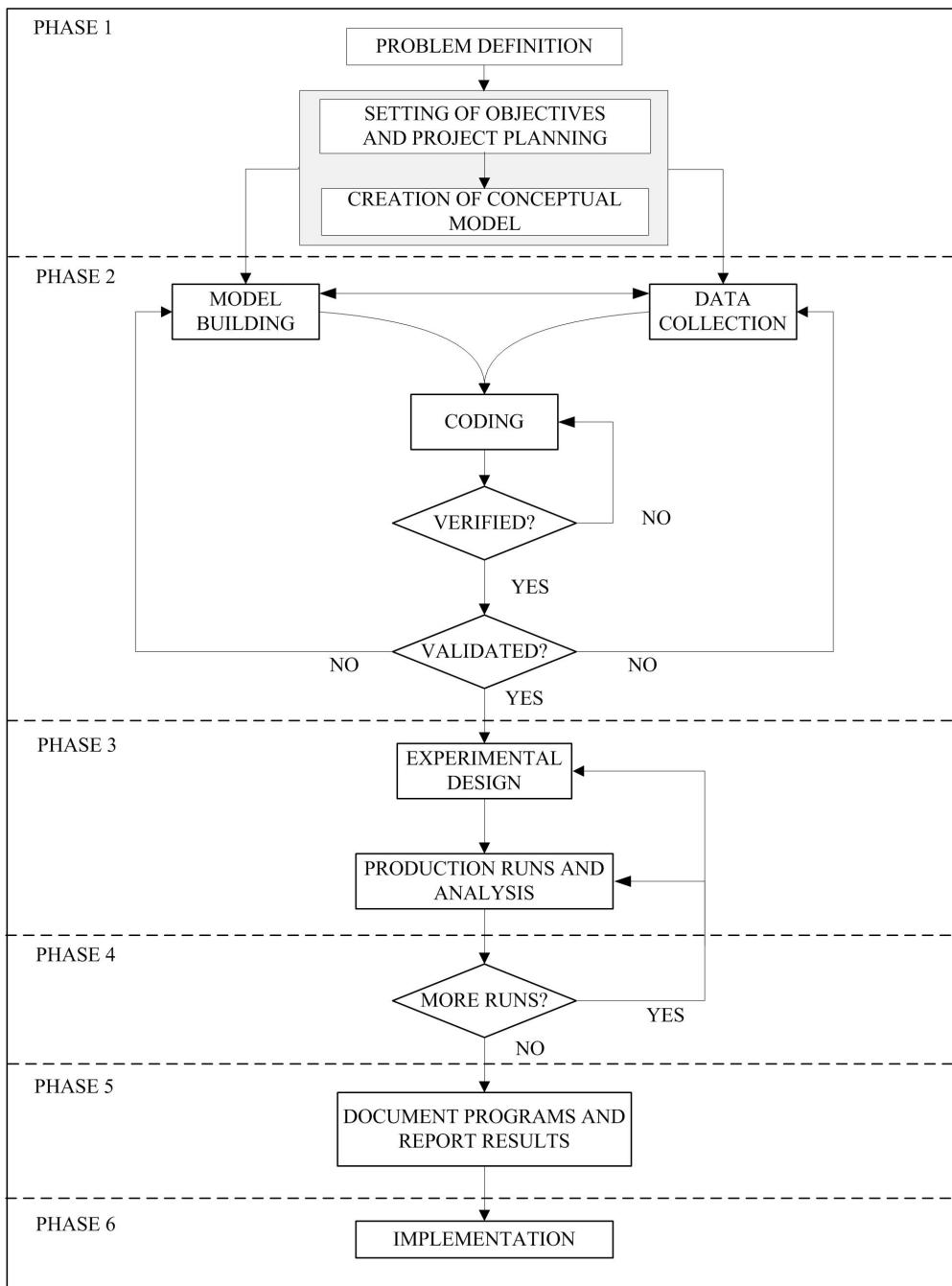


Figure 1.4.: General simulation project methodology

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

1. Simulation Modeling

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 for 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 upon the nature of the problem and the goals of the study. The general goals of a simulation study often include:

- 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 8 represents an excellent sample of problem and system definition statements. If you have the opportunity to do a "real-life" project as part of your study of simulation, you might want to utilize the book problems in this text and the example in Chapter 8 for how to write reasonable problem/system definition statements.

With a good understanding of the problem and of 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:

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, state charts, etc 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

1. Simulation Modeling

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, don't 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:

- Input Data Preparation: Input data is analyzed to determine the nature of the data and to determine 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

1.7. Simulation Methodology

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, 1998).

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 re-run 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, etc) and any advanced output analysis techniques (e.g. batching, initialization bias prevention, variance reduction techniques, multiple comparison procedures, etc.) 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 documenting 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

1. Simulation Modeling

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 will begin our exploration of discrete event simulation and introduce you to the key modeling tool, Arena. Chapter 3 introduces the basic statistical concepts that you will need to begin to understand output generated from a simulation. Chapter 4 presents intermediate process modeling within Arena. After these first four chapters, you will have a solid understanding of how to apply Arena to model many realistic situations via simulation. Then, Chapter 5 returns to a statistical concept within simulation in order to prepare you to analyze the results of a infinite horizon simulation. Chapter 6 presents more advanced process modeling, including an introduction to modeling non-stationary situations. Chapter 7 includes the modeling of material handling and movement of within a system. Finally, Chapter 8, 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.

Along with the basic modeling chapters a number of useful appendices have been provided. Appendix A presents the mathematical basis for random number generation and for generating random variables from probability distributions. Appendix B describes the basic processes for fitting probability distributions. Because simulation often involves the modeling of waiting systems, Appendix C provides an overview of the analytical analysis of single queue systems and many of the important formulas that an analyst may need, especially if they use queueing theory to verify and validate their simulation models. A number of other useful appendices are provided to cover probability distributions, common Arena constructs, and aspects of programming within Arena.

Example models using 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 and attempt 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.

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

1.9. Exercises

Exercise 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.

Exercise 1.2. 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 inter-arrival and service times of the customers necessary to simulate this system.

Exercise 1.3. Classify the systems as either being discrete or continuous:

- Electrical Capacitor (You are interested in modeling the amount of current in a capacitor at any time t).
 - On-line gaming system. (You are interested in modeling the number of people playing Halo 4 at any time t .)
 - An airport. (You are interested in modeling the percentage of flights that depart late on any given day).
-

Exercise 1.4. Classify the systems as either being discrete or continuous:

- Parking lot
 - Level of gas in Fayetteville shale deposit
 - Printed circuit board manufacturing facility
-

Exercise 1.5. Classify the systems as either being discrete or continuous:

1. *Simulation Modeling*

- Classify the systems as either being discrete or continuous:
 - Elevator system (You are interested in modeling the number of people waiting on each floor and traveling within the elevators.)
 - Judicial system (You are interested in modeling the number of cases waiting for trial.)
 - The in-air flight path of an airplane as it moves from an origin to a destination.
-

Exercise 1.6. What is model conceptualization? Give an example of something that might be produced during model conceptualization.

Exercise 1.7. 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: _____.

Exercise 1.8. Which of the following does the problem formulation phase of simulation not include? - Define the system - Establish performance metrics - Verification - Build conceptual models

Exercise 1.9. *Fill in the blank* The general goals of a simulation include the _____ of system alternatives and their performance measures across various factors (decision variables) with respect to some objectives.

Exercise 1.10. *Fill in the blank* The general goals of a simulation include the _____ of system behavior at some future point in time.

Exercise 1.11. *True or False* Verification of the simulation model is performed to determine whether the simulation model adequately represents the real system.

2. Introduction to Simulation and 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 a model of a simple DEDS

In this chapter, we explore the Arena simulation software platform for developing and executing simulation models. After highlighting the Arena modeling environment, we will consider some small models for both static and dynamic simulation. The coverage of static simulation will allow us to introduce the modeling environment without having to introduce the notion of the discrete-event clock. Then, we will begin our study of the major emphasis of this textbook: modeling discrete-event dynamic systems. As defined in Chapter 1, a discrete-event dynamic system (DEDS) is a system that evolves dynamically through time. This chapter will introduce how time evolves for DEDSs and illustrate how to develop a model for a simple queuing system. Let's jump into Arena.

2.1. The Arena Environment

Arena is a commercial software program that facilitates the development and execution of computer simulation models. It 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 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 2.1 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

2. Introduction to Simulation and Arena

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 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 as 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 pre-specified 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 2.2 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 the constructs. We will also use these module names when we write text based pseudo-code to represent our conceptual understanding of the problem to be modeled.

2.2. Performing Simple Monte-Carlo Simulations using Arena

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.

2.2. Performing Simple Monte-Carlo Simulations using Arena

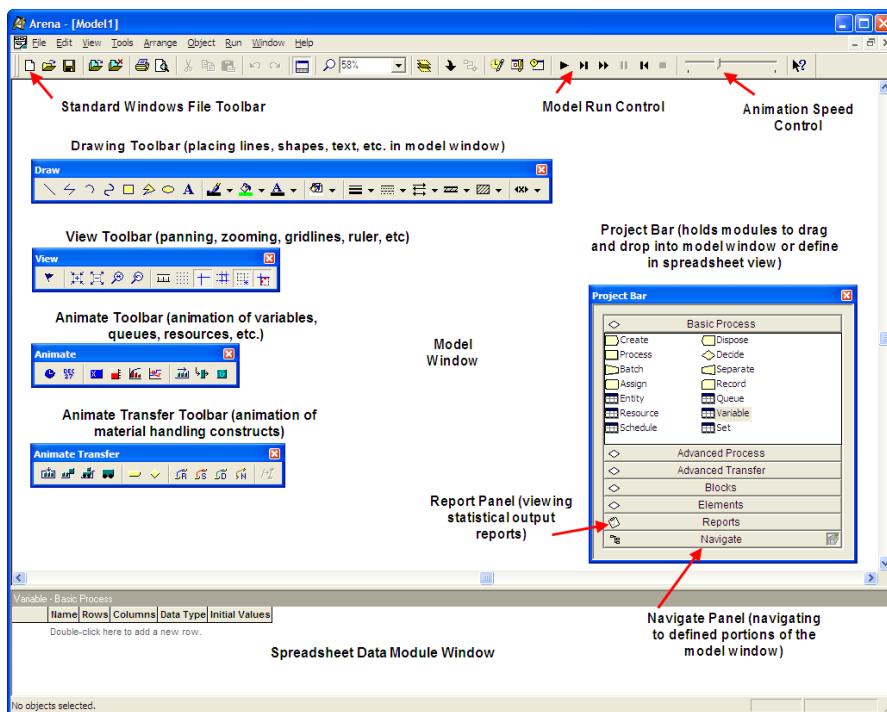


Figure 2.1.: The Arena Environment

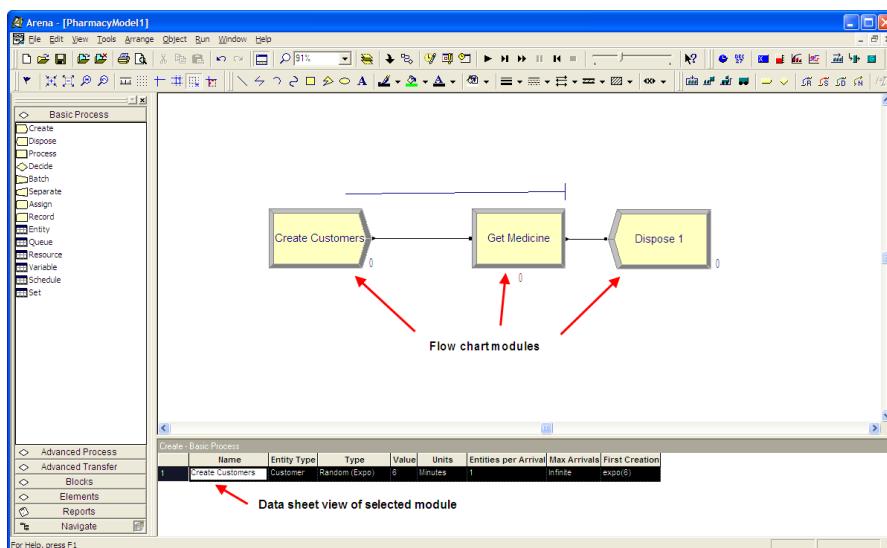


Figure 2.2.: Arena environment with toolbars docked

2. Introduction to Simulation and Arena

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 sections of this chapter and other chapters in the book.

2.2.1. Simple Monte Carlo Integration {ssMC}

In this section, 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:

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

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 $X \sim U(a, b)$:

$$Y = (b - a) g(X) \quad (2.1)$$

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:

$$E[Y] = E[(b - a) g(X)] = (b - a) E[g(X)] \quad (2.2)$$

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:

2.2. Performing Simple Monte-Carlo Simulations using Arena

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

Therefore,

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

Substituting this result into Equation (2.2), yields,

$$\begin{aligned} E[Y] &= E[(b-a)g(X)] = (b-a)E_X[g(X)] \\ &= (b-a) \int_a^b g(x)\frac{1}{b-a}dx \\ &= \int_a^b g(x)dx = \theta \end{aligned} \tag{2.3}$$

Therefore, by estimating the expected value of Y , we can estimate the desired integral. From basic statistics, we know that a good estimator for $E[Y]$ 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 the variable X . Substituting each X_i into Equation (2.1) yields the i^{th} observation of Y ,

$$Y_i = (b-a)g(X_i) \tag{2.4}$$

Then, the sample average of is:

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

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 multi-dimensional integrals. While this discussion is based on a single valued function, the theory scales to multi-dimensional integration through the use of multi-variate distributions. Now, let's illustrate the estimation of the area under a curve for a simple function.

2. Introduction to Simulation and Arena

Example 2.1. 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 2.3. That is, we want to evaluate the integral:

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

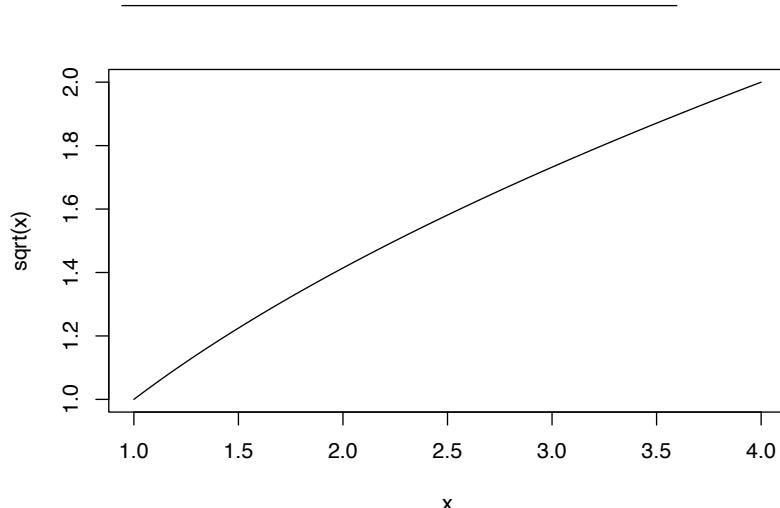


Figure 2.3.: $f(x)$ over the range from 1 to 4

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

Example 2.1 estimates the area under a very simple function. In fact, we know from basic calculus that the area is $\theta = 14/3 = 4.6\bar{6}$. As will be seen in the following solution, the result of the simulation 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 more about this important topic in Section 3.3.

Even though this example is rather simple, the basic idea can be used for any complicated function and can be readily adapted for multi-dimensional integrals. Now, let's solve this problem using Arena.

2.2.2. Arena Modules Needed for Static Simulation Examples

In this section we will implement the Monte Carlo area estimation problem using Arena. First, we need to introduce four programming modules that we will use to develop these simulations.

2.2. Performing Simple Monte-Carlo Simulations using Arena

- 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 to 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 will be illustrated in this section.

Here is a 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.
- 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.

In the following sections, we will put these modules and functions into practice.

2.2.3. Area Estimation via Arena

In the example, we are interested in estimating the area under $f(x) = x^{\frac{1}{2}}$ over the range from 1 to 4. The following pseudo-code outlines the basic code that will be used for estimating the area of the function within Arena.

2. Introduction to Simulation and Arena

```

CREATE 1 entity
BEGIN ASSIGN
    vXi = UNIF(1,4)
    vFx = SQRT(vXi)
    vY = (4.0 - 1.0)*vFx
END ASSIGN
RECORD vY
DISPOSE

```

This pseudo-code is very self-explanatory. It outlines the modeling approach using syntax that is similar to constructs that appear in Arena. We will define more carefully the use of pseudo-code in the modeling process starting in Chapter 3. The CREATE statement generates 1 entity, which then executes the rest of the code. We will learn more about the concept of an entity when we use entities in the DEDS modeling in later sections of the book. 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 model. However, it is useful to conceptualize the program in this fashion so that you can more readily implement it within the Arena environment.

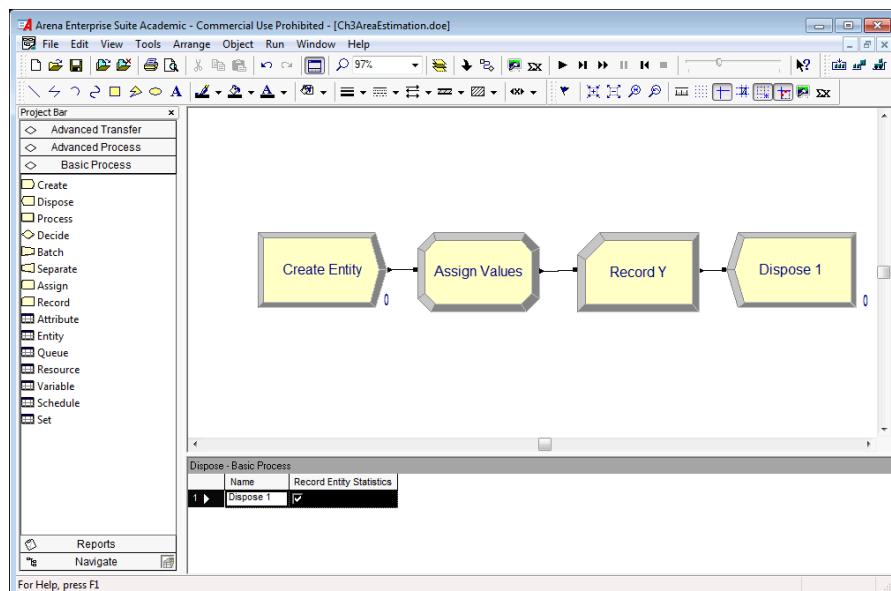


Figure 2.4.: Arena model for estimating the area of $f(x) = \sqrt{x}$

Figure 2.4 represent the flow chart solution within the 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 2.5 illustrates the creating of 1 entity by

2.2. Performing Simple Monte-Carlo Simulations using Arena

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.

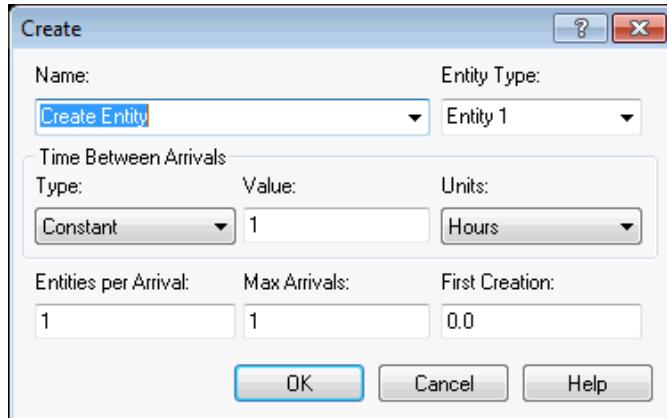


Figure 2.5.: Creating 1 entity with a CREATE module

An ASSIGN module permits the assignment of values to various quantities of interest in the model. In Figure 2.6, 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!

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 2.6.: Making assignments using an ASSIGN module

Figure 2.7 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 output reports, labeled by the text in the Tally Name field. The results are illustrated in Figure 2.8.

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 to run the model. This is accomplished using the Run Setup menu item. Figure 2.9 shows that the model is set to run for 20 replications. Replications will be discussed later in the text, but for now, you can think of a replication as a running of the model to pro-

2. Introduction to Simulation and Arena

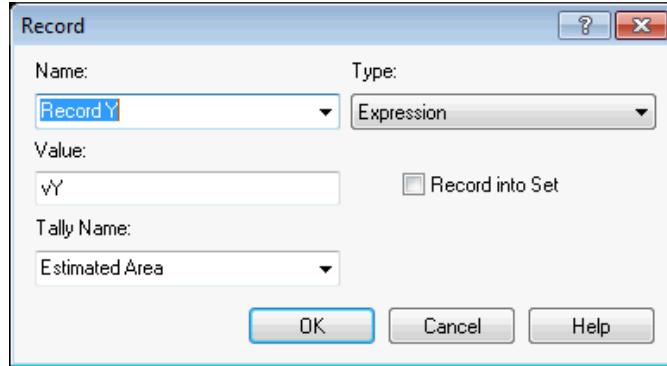


Figure 2.7.: Recording statistics on an expression using a RECORD module

User Specified		
Tally		
Expression	Average	Half Width
Estimated Area	4.3872	0.39

Figure 2.8.: Results from the area estimation model

duce observations. Each replication is independent of other replications. Thus, the observations across replications form a random sample of the observations. Arena will automatically report the statistics across the replications.

Figure 2.8 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 for Example 2.1. The difference between the estimated value and the true value is due to sampling error. We will discuss how to control sampling error within Chapter 3. The area estimation model can be found within this Chapter's files as *AreaEstimation.doe*.

Now let's look at a slightly more complex static simulation.

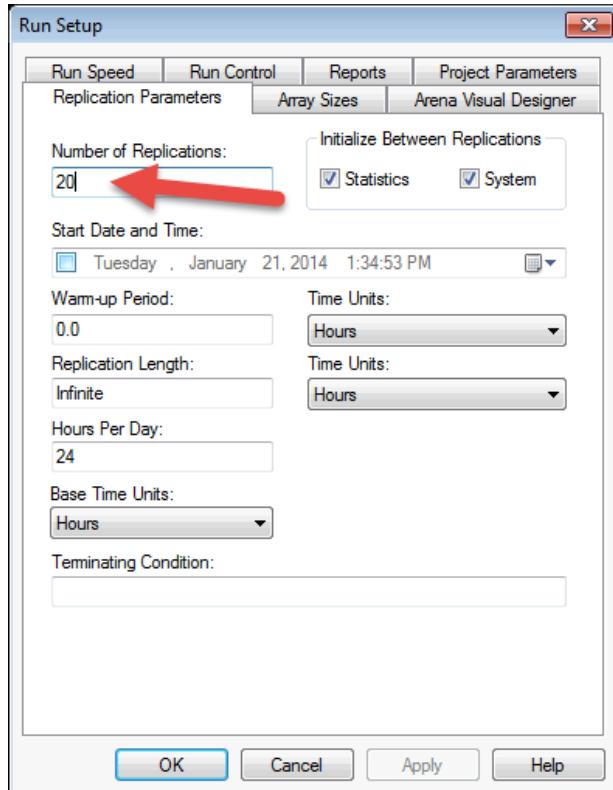


Figure 2.9.: Setting up the simulation to replicate 20 observations

2.2.4. The News Vendor 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 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 covering 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 cumulative distribution function 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

2. Introduction to Simulation and Arena

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 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 left over paper to a recycling center. We assume that $\text{selling price} > \text{purchase cost} > \text{salvage value}$.

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: 1) demand is less than the amount ordered ($D < q$) or 2) 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:

$$G(D, q) = s \times \min(D, q) + u \times \max(0, q - D) - c \times q \quad (2.5)$$

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 (2.5), 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's look at an example.

Example 2.2. Sly's convenience store sells BBQ wings for 25 cents a piece. They cost 15 cents a piece to make. 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 2.1.

Table 2.1.: 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

The pseudo-code for the news vendor problem will require the generation of the demand from BBQ Wing demand distribution. In order to generate from this distribution, the DISC() function should be used. Since the cumulative distribution has already been computed, it is straight forward 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 as follows.

2. Introduction to Simulation and Arena

```

CREATE 1 entity
BEGIN 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

```

The CREATE module creates the 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 previous discussion. The flow chart solution is illustrated in Figure 2.10 and uses the same flow chart modules (CREATE, ASSIGN, RECORD, DISPOSE) as in the last example.

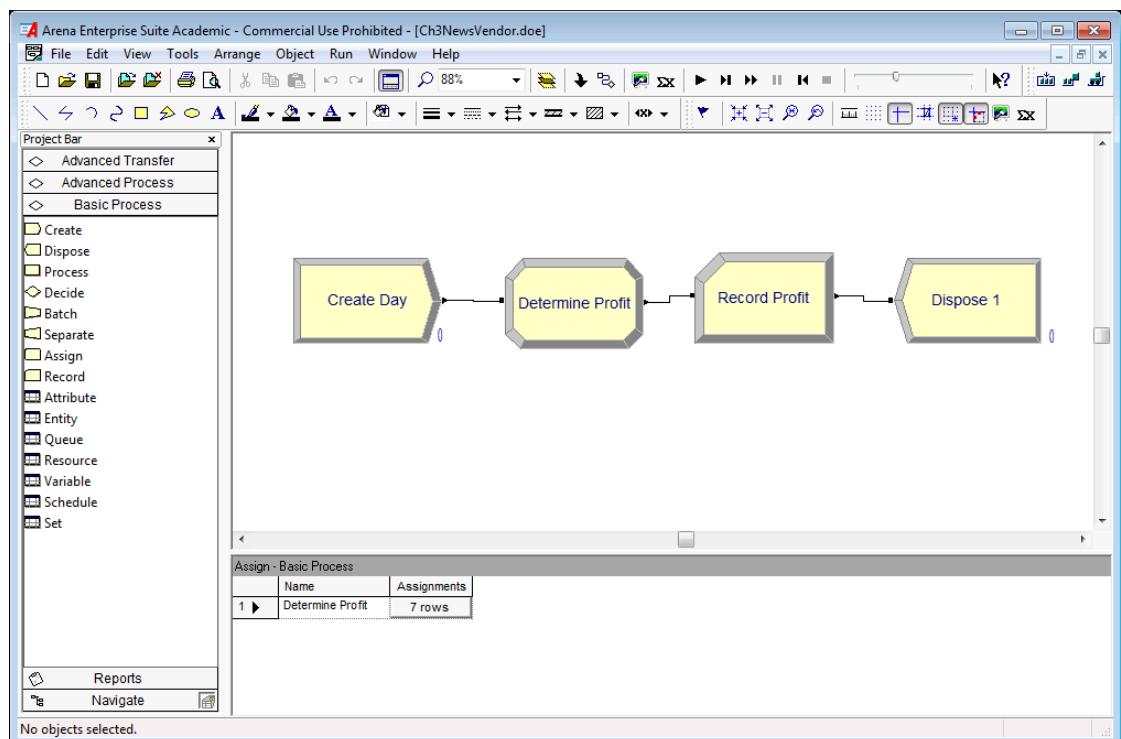


Figure 2.10.: Model for news vendor problem

2.2. Performing Simple Monte-Carlo Simulations using Arena

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 2.11 the definition of each variable and how the initial value of the variable *vCostPerUnit* is showing. The CREATE module is setup exactly as in the last example. The created entity will move to the next module (ASSIGN) illustrated in Figure 2.12. The assignments follow exactly the previously illustrate pseudo-code. Running the model for 100 days results in the output illustrated in Figure 2.13.

Variable - Basic Process							
	Name	Rows	Columns	Data Type	Clear Option	File Name	Initial Values
1 ►	<i>vCostPerUnit</i>			Real	System		1 rows 0.15
2	<i>vPricePerUnit</i>			Real	System		1 rows
3	<i>vSalvagePerUnit</i>			Real	System		1 rows
4	<i>vOrderQty</i>			Real	System		1 rows
5	<i>vDemand</i>			Real	System		0 rows
6	<i>vAmtSold</i>			Real	System		0 rows
7	<i>vAmtLeft</i>			Real	System		0 rows
8	<i>vSalesRevenue</i>			Real	System		0 rows
9	<i>vSalvageRevenue</i>			Real	System		0 rows
10	<i>vOrderingCost</i>			Real	System		0 rows
11	<i>vProfit</i>			Real	System		0 rows

Double-click here to add a new row.

Figure 2.11.: Variable definitions used in the news vendor solution

Assignments			
	Type	Variable Name	New Value
1	Variable	<i>vDemand</i>	DISC(0.1, 5, 0.3, 10, 0.6, 40, 0.8, 45, 0.9, 50, 0.95, 55, 1.0, 60)
2	Variable	<i>vAmtSold</i>	MN(<i>vDemand</i> , <i>vOrderQty</i>)
3	Variable	<i>vAmtLeft</i>	MX(0, <i>vOrderQty</i> - <i>vDemand</i>)
4	Variable	<i>vSalesRevenue</i>	<i>vAmtSold</i> * <i>vPricePerUnit</i>
5	Variable	<i>vSalvageRevenue</i>	<i>vAmtLeft</i> * <i>vSalvagePerUnit</i>
6	Variable	<i>vOrderingCost</i>	<i>vCostPerUnit</i> * <i>vOrderQty</i>
7	Variable	<i>vProfit</i>	<i>vSalesRevenue</i> + <i>vSalvageRevenue</i> - <i>vOrderingCost</i>

Double-click here to add a new row

Figure 2.12.: Assignments used in the news vendor solution

The news vendor model can be found within this Chapter's files as *NewsVendor.doe*.

In this section, we explored how to develop models in 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 estimation problem, time was not a 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, hour 2, etc.). This will be the focus of the rest of the book.

2. Introduction to Simulation and Arena

User Specified		
Tally	Average	Half Width
Record Profit	1.3440	0.47

Figure 2.13.: Results based on 100 replications of the news vendor model

2.3. 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 discrete-event dynamic systems, 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's 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.

Table 2.2.: Customer time of arrivals

Customer	Time of arrival	Time between arrival
1	2	2
2	5	3
3	7	2
4	15	8

Figure 2.14 illustrates a time line of customer arrival events. The values for the arrival times in Table 2.2 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 two and there is an elapse of three minutes 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. An event denotes an instance of time that changes the state of the system. Since an event causes actions that result in a change of system state, it is natural to 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.

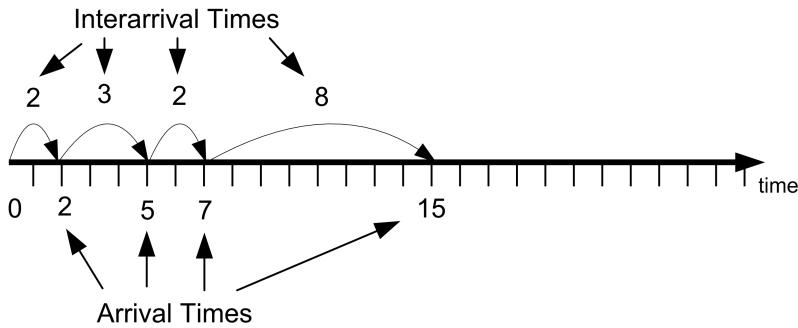


Figure 2.14.: Customer arrival process

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) the number of customers in the bank will decrease by one. 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 2.15 contains the service times for each of the four customers that arrived in Figure 2.14. Examine the figure with respect to customer 1. Based on the figure, customer 1 can enter service at time two 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 as indicated in Table 2.3. From this information, the time that customer 1 is going to complete service can be pre-computed. 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 that 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 2.15 and knowledge of how banks operate should indicate that a customer will begin service either immediately upon

2. Introduction to Simulation and Arena

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 that service completions occur and the times that arrivals occur constitute the pertinent events for simulating this banking system.

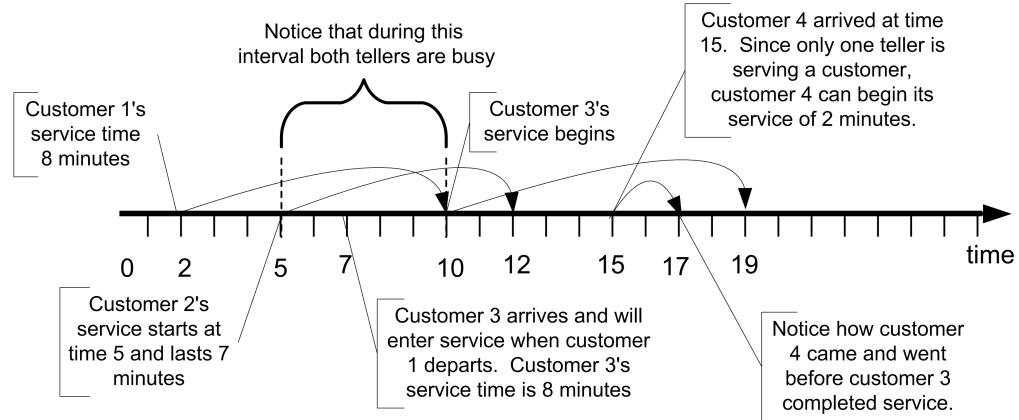


Figure 2.15.: Customer service process

Table 2.3.: Service time for first four customers

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

If the arrival and the service completion events are combined, then the time ordered sequence of events for the system can be determined. Figure 2.16 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 are invoked that change the state of the system.

In the following table, c_i represents the i^{th} customer and s_j represents the j^{th} teller.

Time	Event	Comment
0	Bank Opens	
2	Arrival	c_1 arrives, enters service for 8 min., s_1 becomes busy

Time	Event	Comment
5	Arrival	c_2 arrives, enters service for 7 min., the s_2 becomes busy
7	Arrival	c_3 arrives, waits in queue
10	Service Completion	c_1 completes service, c_3 exits queue, enters service for 9 min.
12	Service Completion	c_2 completes service, queue is empty, s_2 becomes idle
15	Arrival	c_4 arrives, enters service for 2 min., s_2 becomes busy
17	Service Completion	c_4 completes service, s_2 becomes idle
19	Service Completion	c_3 completes service, s_1 becomes idle

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 pre-computed prior to the start of the simulation. For example, at time two, customer 1 arrived. Customer 2 arrives at time 5. Thus, the time between arrivals (3 minutes) is added to the current time 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 a discrete-event dynamic system simulation the 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.

2. Introduction to Simulation and Arena

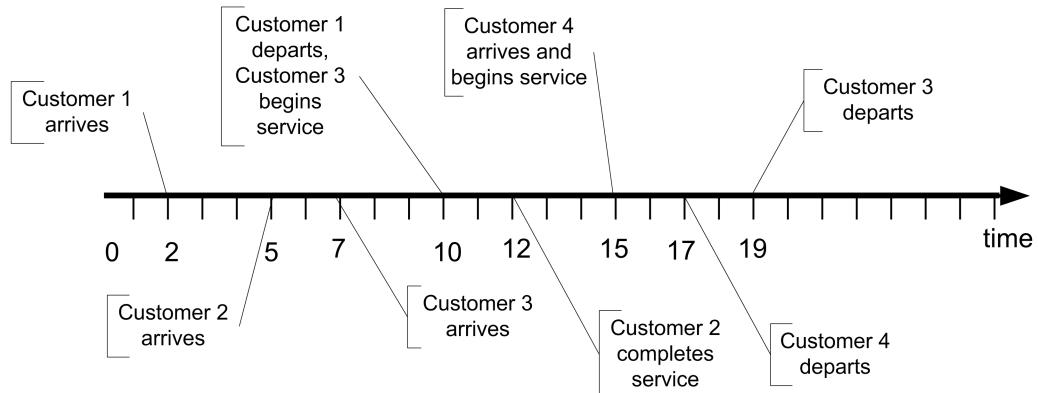


Figure 2.16.: Events ordered by time

2.4. Simulating a Queueing System By Hand

This section builds on the concepts discussed in the previous section in order to provide insights into how discrete event simulations operate. In order to do this, we will simulate a simple queueing system by hand. That is, we will process each of the events that occur as well as the state changes in order to trace the operation of the system through time.

Consider again the simple banking system described in the previous section. To simplify the situation, we assume that there is only one teller that is available to provide service to the arriving customers. Customers that arrive while the teller is already helping a customer form a single waiting line, which is handled in a first come, first served manner. Let's assume that the bank opens at 9 am with no customers present and the teller idle. The time of arrival of the first eight customers is provided in the following Table 2.5

Table 2.5.: Time of arrivals and service times for 8 customers

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

We are going to process these customers in order to recreate the behavior of this system over from time 0 to 31 minutes. To do this, we need to be able to perform the “bookkeeping” that a computer simulation model would normally perform for us. Thus, we will need to define some variables associated with the system and some attributes associated with the customers. Consider the following system variables.

System Variables

- Let t represent the current simulation clock time.
- Let $N(t)$ represent the number of customers in the system (bank) at any time t .
- Let $Q(t)$ represent the number of customers waiting in line for the teller at any time t .
- Let $B(t)$ represent whether or not the teller is busy (1) or idle (0) at any time t .

$$N(t) = Q(t) + B(t)$$

Note also that, knowledge of the value of $N(t)$ is sufficient to determine the values of $Q(t)$ and $B(t)$ at any time t . For example, if we know that there are 3 customers in the bank, then $N(t) = 3$, and since the teller must be serving 1 of those customers, $B(t) = 1$ and there must be 2 customers waiting, $Q(t) = 2$. In this situation, since $N(t)$, is sufficient to determine all the other system variables, we can say that $N(t)$ is the system's *state variable*. The state variable(s) are the minimal set of system variables that are necessary to represent the system's behavior over time. We will keep track of the values of all of the system variables during our processing in order to collect statistics across time.

Attributes are properties of things that move through the system. In the parlance of simulation, we call the things that move through the system entity instances or entities. In this situation, the entities are the customers, and we can define a number of attributes for each customer. Here, customer is a type of entity or entity type. If we have other things that flow through the system, we identify the different types (or entity types). The attributes of the customer entity type are as follows. Notice that each attribute is subscripted by i , which indicates the i^{th} customer instance.

Entity Attributes

- Let ID_i be the identity number of the customer. ID_i is a unique number assigned to each customer that identifies the customer from other customers in the system.
- Let A_i be the arrival time of the i^{th} customer.
- Let S_i be the time the i^{th} customer started service.
- Let D_i be the departure time of the i^{th} customer.
- Let ST_i be the service time of the i^{th} customer.
- Let T_i be the total time spent in the system of the i^{th} customer.
- Let W_i be the time spent waiting in the queue for the i^{th} customer.

Clearly, there are also relationships between these attributes. We can compute the total time in the system for each customer from their arrival and departure times:

$$T_i = D_i - A_i$$

In addition, we can compute the time spent waiting in the queue for each customer as:

$$W_i = T_i - ST_i = S_i - A_i$$

As in the previous section, there are two types of events: arrivals and departures. Let $E(t)$ be (A) for an arrival event at time t and be (D) for a departure event at time t . As we process each

2. Introduction to Simulation and Arena

Table 2.6.: Bank by hand simulation bookkeeping table.

System Variables					Entity Attributes							
t	E(t)	N(t)	B(t)	Q(t)	ID(i)	A(i)	S(i)	ST(i)	D(i)	T(i)	W(i)	Pending E(t)
0	NA	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA

Table 2.7.: Bank by hand simulation bookkeeping table.

System Variables					Entity Attributes							
t	E(t)	N(t)	B(t)	Q(t)	ID(i)	A(i)	S(i)	ST(i)	D(i)	T(i)	W(i)	Pending E(t)
0	NA	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA
3	A	1	1	0	1	3	4	3	NA	NA	0	E(7) = D(1), E(11) = A(2)

event, we will keep track of when the event occurs and the type of event. We will also keep track of the state of the system after each event has been processed. To make this easier, we will keep track of the system variables and the entity attributes within a table as follows.

In the table, we see that the initial state of the system is empty and idle. Since there are no customers in the bank, there are no tabulated attribute values within the table. Reviewing the provided information, we see that customer 1 arrives at $t = 3$ and has a service time of 4 minutes. Thus, $ID_1 = 1$, $A_1 = 3$, and $ST_1 = 4$. We can enter this information directly into our bookkeeping table. In addition, because the bank was empty and idle when the customer arrived, we can note that the time that customer 1 starts service is the same as the time of their arrival and that they did not spend any time waiting in the queue. Thus, $S_1 = 3$ and $W_1 = 0$. The table has been updated as follows.

Because customer 1 arrived to an empty system, they immediately started service at time 3. Since we know the service time of the customer, $ST_1 = 4$, and the current time, $t = 3$, we can determine that customer 1, will depart from the system at time 7 ($t = 3 + 4 = 7$). That means we will have a departure event for customer 1 at time 7. According to the provided data, the next customer, customer 2, will arrive at time 11. Thus, we have two pending events, a departure of customer 1 at time 7 and the arrival of customer 2 at time 11. This fact is noted in the column labeled “Pending E(t)” for pending events. Here $E(7) = D(1)$, $E(11) = A(2)$ indicates that customer 1 with depart, D_1 , at time 7, $E(7)$ and customer 2 will arrive, A_2 , at the event at time 11, $E(11)$. Clearly, the next event time will be the minimum of 7 and 11, which will be the departure of the first customer. Thus, our bookkeeping table can be updated as follows.

Since there are no customers in the bank and only the one pending event, the next event will be the arrival of customer 2 at time 11. The table can be updated as follows.

Since the pending event set is $E(13) = A(3)$, $E(15) = D(2)$ the next event will be the arrival of the third customer at time 13 before the departure of the second customer at time 15. We will now have a queue form. Updating our bookkeeping table, yields:

2.4. Simulating a Queueing System By Hand

Table 2.8.: Bank by hand simulation bookkeeping table.

System Variables					Entity Attributes							
t	E(t)	N(t)	B(t)	Q(t)	ID(i)	A(i)	S(i)	ST(i)	D(i)	T(i)	W(i)	Pending E(t)
0	NA	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA
3	A	1	1	0	1	3	4	3	NA	NA	0	E(7) = D(1), E(11) = A(2)
7	D	0	0	0	1	3	4	3	7	4	0	E(11) = A(2)

Table 2.9.: Bank by hand simulation bookkeeping table.

System Variables					Entity Attributes							
t	E(t)	N(t)	B(t)	Q(t)	ID(i)	A(i)	S(i)	ST(i)	D(i)	T(i)	W(i)	Pending E(t)
0	NA	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA
3	A	1	1	0	1	3	4	3	NA	NA	0	E(7) = D(1), E(11) = A(2)
7	D	0	0	0	1	3	4	3	7	4	0	E(11) = A(2)
11	A	1	1	0	2	11	4	11	NA	NA	0	E(13) = A(3), E(15) = D(2)

Notice that in the last table update, we did not update S_i and W_i because customer 3 had to wait in queue and did not start service. Customer 3 will start service, when customer 2 departs. Reviewing the pending event set, we see that the next event will be the arrival of customer 4 at time 14, which yields the following bookkeeping table.

Notice that we now have 3 customers in the system, 1 in service and 2 waiting in the queue. Reviewing the pending event set, we see that customer 2 will finally complete service and depart at time 15.

Because customer 2 completes service at time 15 and customer 3 is waiting in the line, we see that customer 3's attributes for S_i and W_i were updated within the table. Since customer 3 has started service and we know their service time of 4 minutes, we know that they will depart at time 19. The pending events set has been updated accordingly and indicates that the next event will be the arrival of customer 5 at time 17.

Table 2.10.: Bank by hand simulation bookkeeping table.

System Variables					Entity Attributes							
t	E(t)	N(t)	B(t)	Q(t)	ID(i)	A(i)	S(i)	ST(i)	D(i)	T(i)	W(i)	Pending E(t)
0	NA	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA
3	A	1	1	0	1	3	4	3	NA	NA	0	E(7) = D(1), E(11) = A(2)
7	D	0	0	0	1	3	4	3	7	4	0	E(11) = A(2)
11	A	1	1	0	2	11	4	11	NA	NA	0	E(13) = A(3), E(15) = D(2)
13	A	2	1	1	3	13	4	15	NA	NA	2	E(14) = A(4), E(15) = D(2)

2. Introduction to Simulation and Arena

Table 2.11.: Bank by hand simulation bookkeeping table.

System Variables					Entity Attributes							
t	E(t)	N(t)	B(t)	Q(t)	ID(i)	A(i)	S(i)	ST(i)	D(i)	T(i)	W(i)	Pending E(t)
0	NA	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA
3	A	1	1	0	1	3	4	3	NA	NA	0	E(7) = D(1), E(11) = A(2)
7	D	0	0	0	1	3	4	3	7	4	0	E(11) = A(2)
11	A	1	1	0	2	11	4	11	NA	NA	0	E(13) = A(3), E(15) = D(2)
13	A	2	1	1	3	13	4	15	NA	NA	2	E(14) = A(4), E(15) = D(2)
14	A	3	1	2	4	14	3	19	NA	NA	5	E(15) = D(2), E(17) = A(5)

Table 2.12.: Bank by hand simulation bookkeeping table.

System Variables					Entity Attributes							
t	E(t)	N(t)	B(t)	Q(t)	ID(i)	A(i)	S(i)	ST(i)	D(i)	T(i)	W(i)	Pending E(t)
0	NA	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA
3	A	1	1	0	1	3	4	3	NA	NA	0	E(7) = D(1), E(11) = A(2)
7	D	0	0	0	1	3	4	3	7	4	0	E(11) = A(2)
11	A	1	1	0	2	11	4	11	NA	NA	0	E(13) = A(3), E(15) = D(2)
13	A	2	1	1	3	13	4	15	NA	NA	2	E(14) = A(4), E(15) = D(2)
14	A	3	1	2	4	14	3	19	NA	NA	5	E(15) = D(2), E(17) = A(5)
15	D	2	1	1	2	11	4	11	15	4	0	E(17) = A(5), E(19) = D(3)

Table 2.13.: Bank by hand simulation bookkeeping table.

System Variables					Entity Attributes							
t	E(t)	N(t)	B(t)	Q(t)	ID(i)	A(i)	S(i)	ST(i)	D(i)	T(i)	W(i)	Pending E(t)
0	NA	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA
3	A	1	1	0	1	3	4	3	NA	NA	0	E(7) = D(1), E(11) = A(2)
7	D	0	0	0	1	3	4	3	7	4	0	E(11) = A(2)
11	A	1	1	0	2	11	4	11	NA	NA	0	E(13) = A(3), E(15) = D(2)
13	A	2	1	1	3	13	4	15	NA	NA	2	E(14) = A(4), E(15) = D(2)
14	A	3	1	2	4	14	3	19	NA	NA	5	E(15) = D(2), E(17) = A(5)
15	D	2	1	1	2	11	4	11	15	4	0	E(17) = A(5), E(19) = D(3)
17	A	3	1	2	5	17	2	22	NA	NA	5	E(19) = D(3), E(19) = A(6)

2.4. Simulating a Queueing System By Hand

Table 2.14.: Bank by hand simulation bookkeeping table.

System Variables					Entity Attributes							
t	E(t)	N(t)	B(t)	Q(t)	ID(i)	A(i)	S(i)	ST(i)	D(i)	T(i)	W(i)	Pending E(t)
0	NA	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA
3	A	1	1	0	1	3	4	3	NA	NA	0	E(7) = D(1), E(11) = A(2)
7	D	0	0	0	1	3	4	3	7	4	0	E(11) = A(2)
11	A	1	1	0	2	11	4	11	NA	NA	0	E(13) = A(3), E(15) = D(2)
13	A	2	1	1	3	13	4	15	NA	NA	2	E(14) = A(4), E(15) = D(2)
14	A	3	1	2	4	14	3	19	NA	NA	5	E(15) = D(2), E(17) = A(5)
15	D	2	1	1	2	11	4	11	15	4	0	E(17) = A(5), E(19) = D(3)
17	A	3	1	2	5	17	2	22	NA	NA	5	E(19) = D(3), E(19) = A(6)
19	D	2	1	1	3	13	4	15	19	6	2	E(19) = A(6)
19	A	3	1	2	6	19	4	24	NA	NA	5	E(21) = A(7), E(22) = D(4)
21	A	4	1	3	7	21	3	28	NA	NA	7	E(22) = D(4), E(24) = D(5)
22	D	3	1	2	4	14	3	19	22	8	5	E(24) = D(5), E(27) = A(8)
24	D	2	1	1	5	17	2	22	24	7	5	E(27) = A(8), E(28) = D(6)
27	A	3	1	2	8	27	2	31	NA	NA	4	E(28) = D(6), E(31) = D(7)
28	D	2	1	1	6	19	4	24	28	9	5	E(31) = D(7)
31	D	1	1	0	7	21	3	28	31	10	7	E(33) = D(8)

Now, we have a very interesting situation with the pending event set. We have two events that are scheduled to occur at the same time, the departure of customer 3 at time 19 and the arrival of customer 6 at time 19. In general, the order in which events are processed that occur at the same time may affect how future events are processed. That is, the order of event processing may change the behavior of the system over time and thus the order of processing is, in general, important. However, in this particular simple system, the order of processing will not change what happens in the future. We will simply have an update of the state variables at the same time. In this instance, we will process the departure event first and then the arrival event. If you are not convinced that this will not make a difference, then I encourage you to change the ordering and continue the processing. In more complex system simulations, a priority indicator is attached to the events so that the events can be processed in a consistent manner. Rather than continuing the step-by-step processing of the events through time 31, we will present the completed table. We leave it as an exercise for the reader to continue the processing of the customers. The completed bookkeeping table at time 31 is as follows.

Given that we have simulated the bank over the time frame from 0 to 31 minutes, we can now compute performance statistics for the bank and measure how well it is operating. Two statistics that we can easily compute are the average time spent waiting in line and the average time spent in the system.

Let n be the number of customers observed to depart during the simulation. In this simulation, we had $n = 7$ customers depart during the time frame of 0 to 31 minutes. The time frame

2. Introduction to Simulation and Arena

over which we analyze the performance of the system is called the simulation time horizon. In this case, the simulation time horizon is fixed and known in advance. When we perform a computer simulation experiment, the time horizon is often referred to as the *simulation run length* (or *simulation replication length*). To estimate the average time spent waiting in line and the average time spent in the system, we can simply compute the sample averages (\bar{T} and \bar{W}) of the observed quantities (T_i and W_i) for each departed customer.

$$\bar{T} = \frac{1}{7} \sum_{i=1}^7 T_i = \frac{4 + 4 + 6 + 8 + 7 + 9 + 10}{7} = \frac{48}{7} \cong 6.8571$$

$$\bar{W} = \frac{1}{7} \sum_{i=1}^7 W_i = \frac{0 + 0 + 2 + 5 + 5 + 5 + 7}{7} = \frac{24}{7} \cong 3.4286$$

Besides the average time spent in the system and the average time spent waiting, we might also want to compute the average number of customers in the system, the average number of customers in the queue, and the average number of busy tellers. You might be tempted to simply average the values in the $N(t)$, $B(t)$, and $Q(t)$ columns of the bookkeeping table. Unfortunately, that will result in an incorrect estimate of these values because a simple average will not take into account how long each of the variables persisted with its values over time. In this situation, we are really interested in computed a time average. This is because the variables, $N(t)$, $B(t)$, and $Q(t)$ are time-based variables. This type of variable is always encountered in discrete event simulations.

To make the discussion concrete, let's focus on the number of customers in the queue, $Q(t)$. Notice the number of customers in the queue, $Q(t)$ takes on constant values during intervals of time corresponding to when the queue has a certain number of customers. $Q(t) = \{0, 1, 2, 3, \dots\}$. The values of $Q(t)$ form a step function in this particular case. The following figure illustrates the step function nature of this type of variable. A realization of the values of variable is called a sample path.

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, $f(\bullet)$, continuous on an interval (a, b) , there exists a constant, c , such that

$$\int_a^b f(x) dx = f(c)(b - a)$$

$$f(c) = \frac{\int_a^b f(x) dx}{(b - a)}$$

The value, $f(c)$, is called the mean value of the function. A similar function can be defined for $Q(t)$. This function is called the time-average (and represents the *mean value* of the $Q(t)$ function):

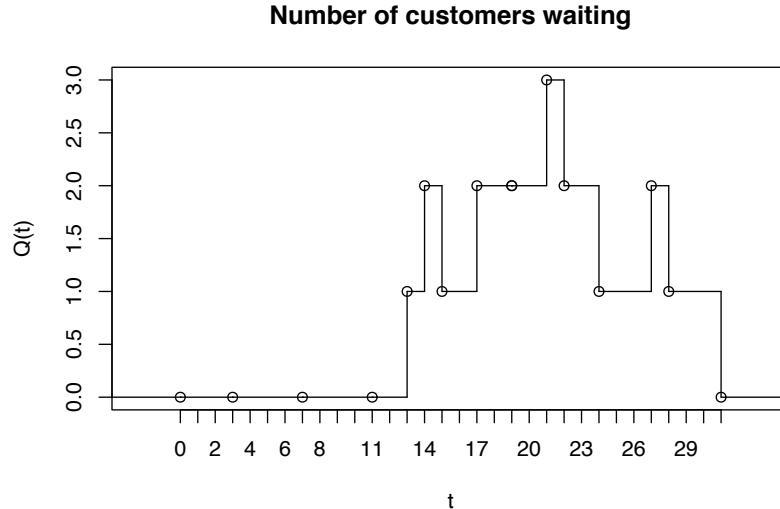


Figure 2.17.: Number of customers waiting in queue for the bank simulation

$$\bar{Q}(n) = \frac{\int_{t_0}^{t_n} Q(t) dt}{t_n - t_0}$$

This function represents the *average with respect to time* of the given state variable. This type of statistical variable is called time-based because $Q(t)$ is a function of 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 $t_{k-1} \leq t \leq t_k$. Then, the time-average can be rewritten as follows:

$$\bar{Q}(t) = \frac{\int_{t_0}^{t_n} Q(t) dt}{t_n - t_0} = \sum_{k=1}^n \frac{q_k(t_k - t_{k-1})}{t_n - t_0}$$

Note that $q_k(t_k - t_{k-1})$ is the area under the curve, $Q(t)$ over the interval $t_{k-1} \leq t \leq t_k$ and because

$$t_n - t_0 = (t_1 - t_0) + (t_2 - t_1) + (t_3 - t_2) + \dots + (t_{n-1} - t_{n-2}) + (t_n - t_{n-1})$$

we can write,

$$t_n - t_0 = \sum_{k=1}^n t_k - t_{k-1}$$

The quantity $t_n - t_0$ represents 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. If we define, $w_k = (t_k - t_{k-1})$, then we can re-write the time average as:

2. Introduction to Simulation and Arena

$$\overline{Q}(t) = \frac{\int_{t_0}^{t_n} Q(t) dt}{t_n - t_0} = \sum_{k=1}^n \frac{q_k(t_k - t_{k-1})}{t_n - t_0} = \frac{\sum_{k=1}^n q_k w_k}{\sum_{i=1}^n w_k}$$

This form of the equation clearly shows that each value of q_k is weighted by:

$$\frac{w_k}{\sum_{i=1}^n w_k} = \frac{w_k}{t_n - t_0} = \frac{(t_k - t_{k-1})}{t_n - t_0}$$

This is why the time average is often called the time-weighted average. If $w_k = 1$, then the time-weighted average is the same as the sample average.

Now we can compute the time average for $Q(t)$, $N(t)$ and $B(t)$. Using the following formula and noting that $t_n - t_0 = 31$

$$\overline{Q}(t) = \sum_{k=1}^n \frac{q_k(t_k - t_{k-1})}{t_n - t_0}$$

We have that the numerator computes as follows:

$$\begin{aligned} \sum_{k=1}^n q_k(t_k - t_{k-1}) &= 0(13 - 0) + 1(14 - 13) + 2(15 - 14) + 1(17 - 15) + 2(19 - 17) \\ &\quad + 1(19 - 19) + 2(21 - 19) + 3(22 - 21) + 2(24 - 22) + \\ &\quad 1(27 - 24) + 2(28 - 27) + 1(31 - 28) = 28 \end{aligned}$$

And, the final time-weighted average number in the queue is:

$$\overline{Q}(t) = \frac{28}{31} \cong 0.903$$

The average number in the system and the average number of busy tellers can also be computed in a similar manner, resulting in:

$$\overline{N}(t) = \frac{52}{31} \cong 1.677$$

$$\overline{B}(t) = \frac{24}{31} \cong 0.7742$$

The value of $\overline{B}(t)$ is most interesting for this situation. Because there is only 1 teller, the fraction of the tellers that are busy is 0.7742. This quantity represents the *utilization* of the teller. The

utilization of a resource represents the proportion of time that the resource is busy. Let c represent the number of units of a resource that are available. Then the utilization of the resource is defined as:

$$\bar{U} = \frac{\bar{B}(t)}{c} = \frac{\int_{t_0}^{t_n} B(t) dt}{c(t_n - t_0)}$$

Notice that the numerator of this equation is simply the total time that the resource is busy. So, we are computing the total time that the resource is busy divided by the total time that the resource could be busy, $c(t_n - t_0)$, which is considered the utilization.

In the following section, we will explore the basic elements of a process-oriented simulation.

2.5. Elements of Process-Oriented Simulation

Chapter 1 described a system as a set of inter-related 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 bank teller example, the service of the customer by the teller 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 inter-related 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 Type A class of things (objects) that interact with the system elements. Entity types describe entity instances.

2. Introduction to Simulation and Arena

Entity or Entity Instance An object of interest in the system whose movement or operation within the system may cause the occurrence of events. Entities are instances of an Entity Type.

Attribute A named property or variable that is associated with an entity type. The value of the attribute is associated with the entity instance described by the entity type.

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.

2.5.1. Entities, Attributes, and Variables

When modeling a system, there are often many entity types. For example, consider a retail store. Besides customers, the products might also be considered as entity types. The products (instances of the entity type) are received by the store and wait on the shelves until customers select them for purchase. Entities (entity instances) 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 type, 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)

2.5. Elements of Process-Oriented Simulation

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 entity types. The values of the attributes for particular entity instances 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 entity types.

Not all information in a system is local to the entity types. 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 entity instances can have access. By making these quantities visible at the system level, the information can be shared between the entity instances within the model and between the components of the system.

Figure 2.18 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, number of busy forklifts, etc. This type of information belongs to the whole system.

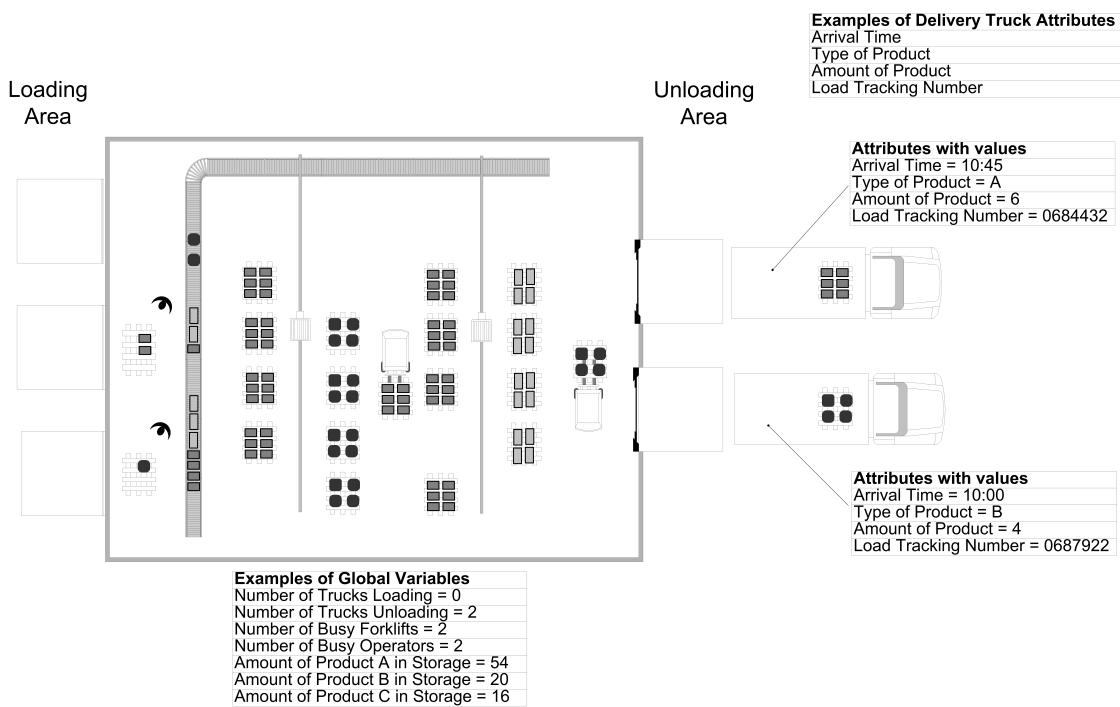


Figure 2.18.: Global variables and attributes within a system

Once a basic understanding of the system is accomplished through understanding system vari-

2. Introduction to Simulation and Arena

ables, 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 is 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 allows the modeler to create and dispose of entities.

2.5.2. Creating and Disposing of Entities

The basic mechanism by which entity instances are introduced into an model is the CREATE module. An entity (entity instance) 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, nothing will happen in a model unless entities are created. 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 2.19 illustrates the CREATE module dialog box that opens when double-clicking on the flow chart symbol within the model window. Understanding what the dialog box entries mean is essential to writing 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.

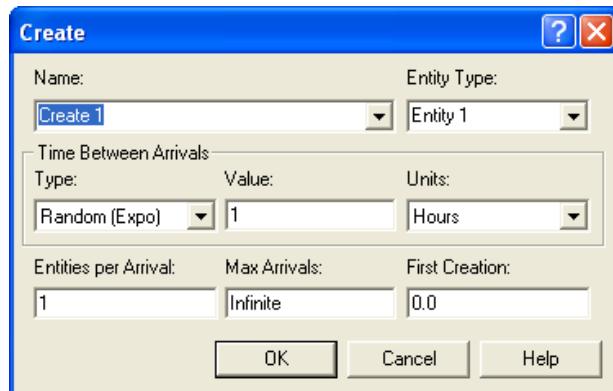


Figure 2.19.: CREATE module dialog box

According to the 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 non-homogeneous Poisson process with rates determined from the specified Schedule

module), *Constant* (user specifies a constant value, e.g., 100), or *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 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 2.20 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.

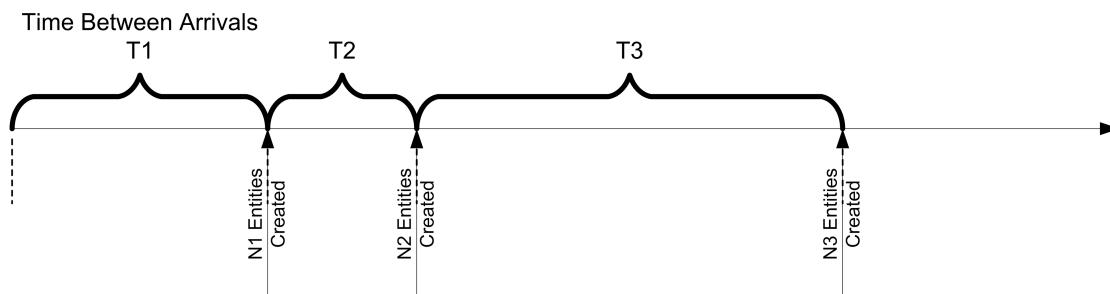


Figure 2.20.: Example arrival process

A CREATE module works by scheduling a future event to occur according to the specified pattern. At each occurrence of the event, a new instance of the entity type is created. The instance is called an entity. The entity instance then executes additional modules until the entity instance hits a module that blocks its progress. More details of how this process works is provided in Section 2.10. The CREATE module continues scheduling the next creation event until the maximum number of creation events is reached or the simulation ends. A common creation process is the Poisson arrival process.

2. Introduction to Simulation and Arena

To specify a Poisson arrival process with mean rate λ , a CREATE module as shown in Figure 2.21 can be used. Why does this specify a Poisson arrival process? Because the time between arrivals for a Poisson process with rate λ is exponentially distributed with the mean of the exponential distribution being $1/\lambda$.

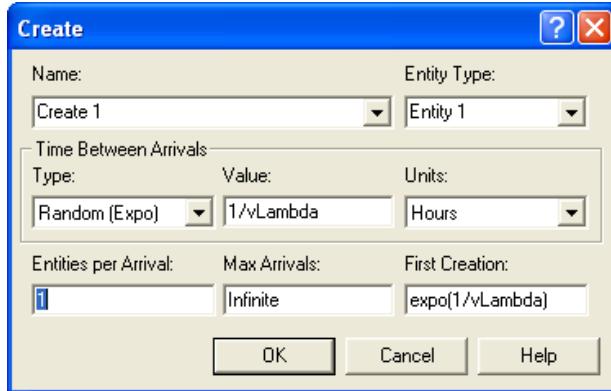


Figure 2.21.: CREATE module for Poisson process

To specify a compound arrival process, use the “Entities per Arrival” field. For example, suppose you have a compound Poisson process where the distribution for the number created at each arrival is governed by a discrete distribution. See (Ross, 1997) for more information on the theory of compound Poisson processes.

$$P(X = x) = \begin{cases} 0.2 & x = 1 \\ 0.3 & x = 2 \\ 0.5 & x = 3 \end{cases}$$

Figure 2.22 shows the CREATE module for this compound Poisson process with the entities per arrival field using the $\text{DISC}(0.2, 1, 0.5, 2, 1.0, 3)$ 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 zero, 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 2.23 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

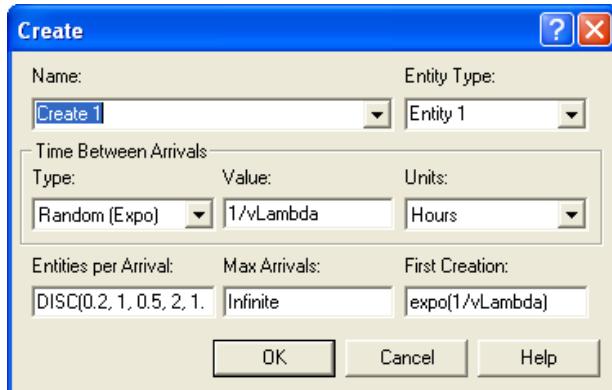


Figure 2.22.: CREATE module for compound Poisson process

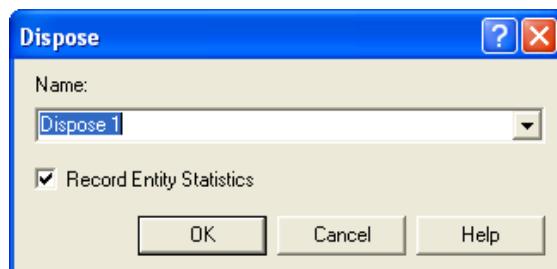


Figure 2.23.: DISPOSE module dialog box

entity statistics will include VA Time (value added time), NVA Time (non-value 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 (non-value 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. These cost allocations assume an activity based costing model which is beyond the scope of this chapter.

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.

2.5.3. 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;*

2. Introduction to Simulation and Arena

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 , 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 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 constructs when reviewing, debugging, and documenting the model.

In addition, the naming rules allow the use of spaces within a variable name, e.g. *This is a legal name* is a legally named variable. 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 *vPart-Count*, 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 *Variables Guide*, which comes with the 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. The VARIABLE module is a data module and is accessed either through the spreadsheet view (Figure 2.24) or through a dialog box (Figure 2.25).

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 2.24.: Data sheet view of VARIABLE data module

Variables can be scalars or can be defined as arrays. The arrays can be either 1-dimensional or 2-dimensional. For 1-dimensional arrays, you specify either the number of rows or the number of columns. There is no difference between an array specified with, say 3 rows, or an array specified with 3 columns. The net effect is that there will be three variables defined which can be indexed

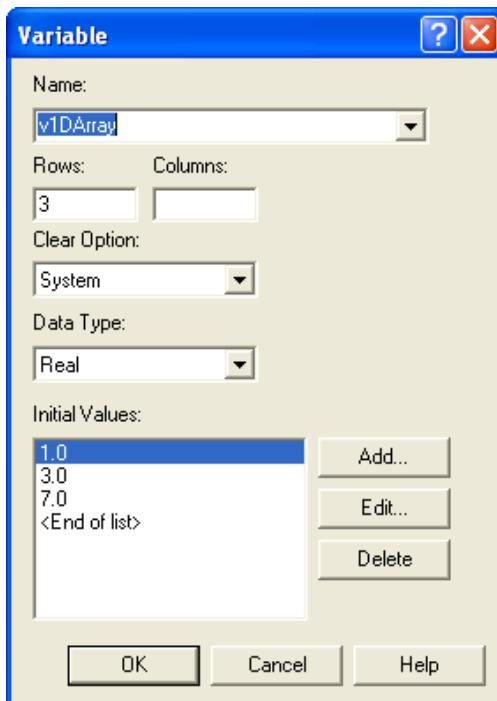


Figure 2.25.: Dialog box for defining variables

by the numbers 1, 2, and 3. When defining a 2-dimensional array of variables (see Figure 2.26), 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.

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
2	v1DArray	3		Real	System	3 rows	<input type="checkbox"/> enter values
3	v2DArray	2	5	Real	System	10 rows	<input type="checkbox"/>

Double-click here to add a new row.

Figure 2.26.: Data sheet view for 2-D variable array

When defining a variable or an array of variables, you may 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 2.24, you can easily enter the initial values for variables and arrays. All variables are treated as real numbers. If you need to represent an integer, e.g. *vPartCount*, you simply repre-

2. Introduction to Simulation and Arena

sent it as a variable. There is no specific integer data type.

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 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 . When an instance of an entity is created within , 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 2.15.: Entities conceptualized as records

IDENT	Entity.SerialNumber	Size	Weight	ProcessingTime
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 2.15 presents six entities conceptualized as records in a table. You can think of the creation and disposing of entities as adding and deleting records within the model's "entity table". The column IDENT represents the IDENT attribute, which uniquely identifies each entity (instance). IDENT is a special pre-defined attribute that is used to uniquely track entities currently in the model. The IDENT attribute is assigned when the entity is created. When an entity is created, 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. Think of IDENT as the primary key into the entity record table.

Every entity in the model has a unique row in the table to store its attribute values. *Entity.SerialNumber* is just one of those attributes. The *Entity.SerialNumber* attribute is also a number assigned to an entity instance 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. When an entity is cloned (duplicated) all attributes *except* IDENT are duplicated. Entity.SerialNumber will not be unique when entities are cloned. IDENT can be used to uniquely identify an entity. Entity.SerialNumber can be used to identify entities that have the same serial number. In other words, all duplicates of an entity instance. 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 pre-defined entity attributes, do a search on "Attributes and Entity-Related Variables" in the Help system.

When you define a user-defined attribute, you are adding another "column" to the "entity table". The implication of this statement is very important. The defining of a user-defined attribute associates that attribute with *every* entity type. This is quite different than what you see in object-oriented languages where you can associate specific attributes with specific classes (types) of objects. Within Arena, an entity type is an entity type. You can specify attributes that can allow you to conceptualize them as being different. 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.

Table 2.16.: Different types of entities conceptualized as records

IDENT	Type	Size	Weight	MoveTime	ProcessingTime
1	1	2	33	—	20
2	1	3	22	—	55
3	2	1	11	23	—

In Table 2.16, 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 instance and then to use the attributes and their values appropriately within that context.

There are two ways to distinguish between different types of entities. Each entity has a pre-

2. Introduction to Simulation and Arena

defined 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 on-line 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 2.27 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 is discussed in Section D.4.

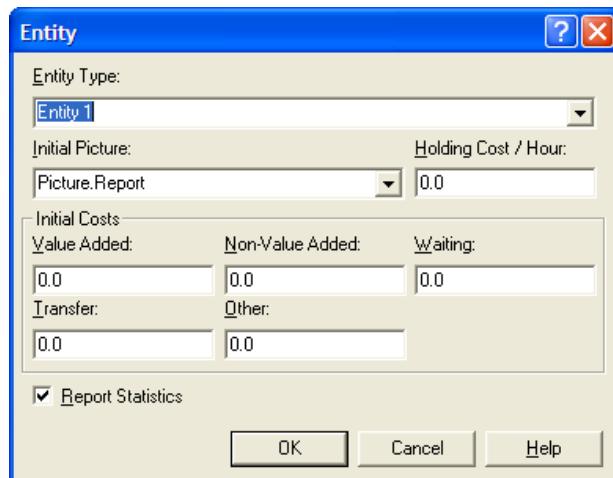


Figure 2.27.: Entity module dialog box

The advantage of using the *Entity.Type* attribute is that some automated functions become easier, such as collecting statistics by entity type (via the DISPOSE module), activity based costing, 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 in a later chapter.

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 2.16.

Now that 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 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 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. Using the ATTRIBUTES module is the preferred method for defining attributes because comments can be provided for the attribute and pre-defining attributes facilitates their selection within drop down text fields or when using the expression builder. This approach tends to avoid annoying syntax errors.

2.6. 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.

2.6.1. 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 2.28. 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 the pharmacist is not too busy.

2.6.2. Modeling the System

Process modeling 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 2.28. Now you should consider the

2. Introduction to Simulation and Arena

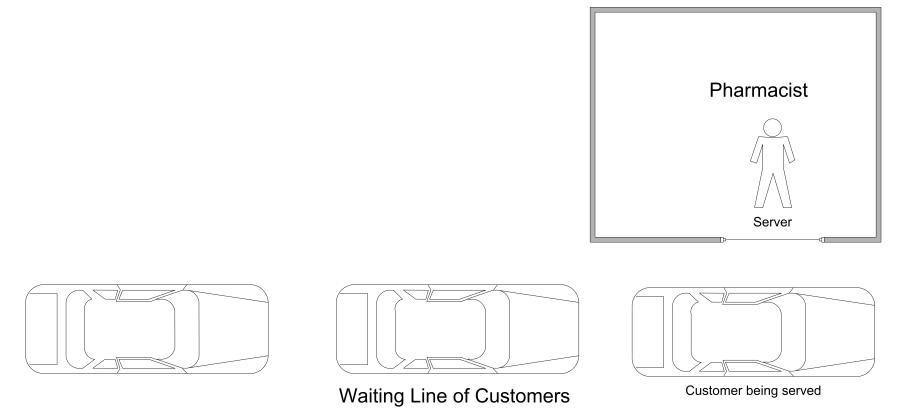


Figure 2.28.: Simple drive through pharmacy

entities 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 a model is: *What are the entities?* In this situation, the entities are the customers that 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 we are modeling 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 an 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 a simulation 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 2.28. 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 . 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), 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 .

Figure 2.29 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, but I recommend just drawing activity diagrams free-hand.

An activity 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 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

2. Introduction to Simulation and Arena

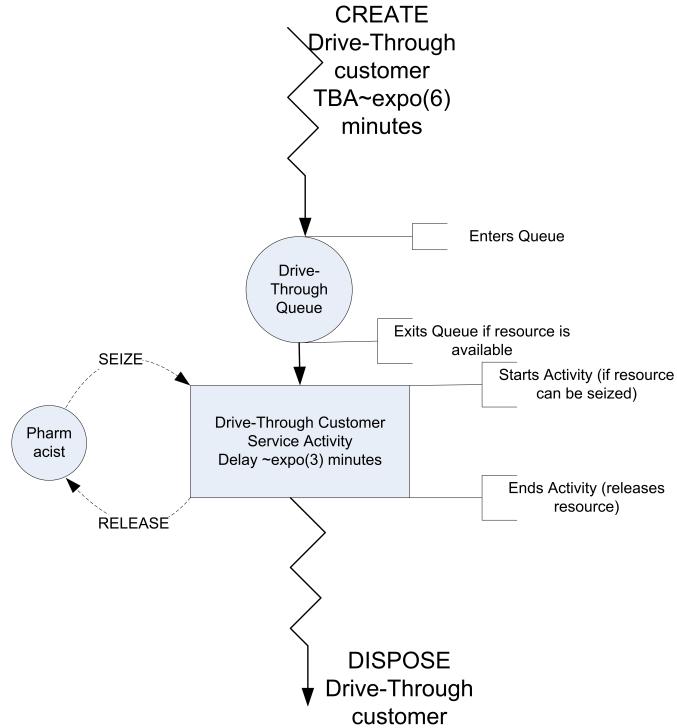


Figure 2.29.: Activity diagram for drive through pharmacy

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 its service activity, the customer leaves the system. This is indicated with the zigzag lines going to no-where and augmented with the keyword DISPOSE. The dashed arrows of a typical activity diagram have also been augmented with the like pseudo-code of SEIZE and RELEASE. The conceptual model of this system can be summarized as follows:

System The system has a pharmacist that acts as a resource, customers that act as entities, and a queue to hold the waiting customers. The state of the system includes the number of customers in the system, in the queue, and in service.

Events Arrivals of customers to the system, which occur within an inter-event 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.

2.6. Modeling a Simple Discrete-Event Dynamic System

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.

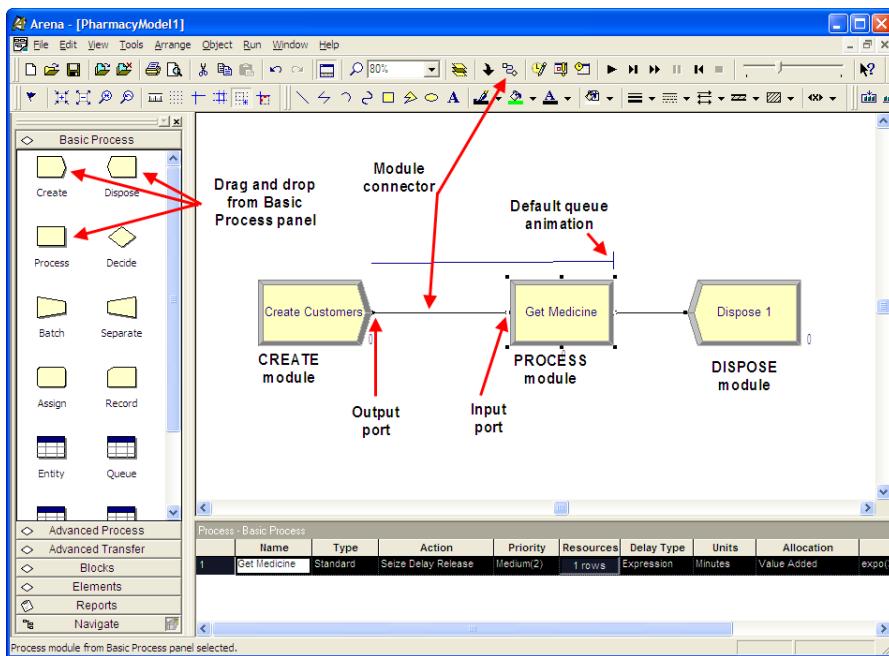


Figure 2.30.: Overview of pharmacy model

2.6.3. Pharmacy Model Implementation

Now, you are ready to implement the conceptual model. If you haven't already done so, open up the Arena Environment. Using the Basic Process Panel template, drag the CREATE, PROCESS, and DISPOSE modules into the model window and make sure that they are connected as shown in Figure 2.30. 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 weren't 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.

2.6.4. Specify the Arrival Process

Within Arena nothing happens unless entities enter the model. This is done through the use of the CREATE module. Actually, this statement is not quite true and not quite false. You don't 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. 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.

$$\frac{1}{\lambda} = \frac{1 \text{ hour}}{10 \text{ customers}} \times \frac{60 \text{ minutes}}{1 \text{ hour}} = \frac{6 \text{ minutes}}{\text{customer}} \quad (2.6)$$



Do not make the mistake of using the arrival rate within the CREATE module The CREATE module uses the time between arrivals. Convert your arrival rate to the time between arrivals as shown in Equation (2.6).

Open up the CREATE module (by double clicking on it) and fill it in as shown in Figure 2.31. The distribution for the "Time Between Arrivals" is by default Exponential, Random (expo) in the figure. In this instance, the "Value" textbox 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 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 in 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 2.32.

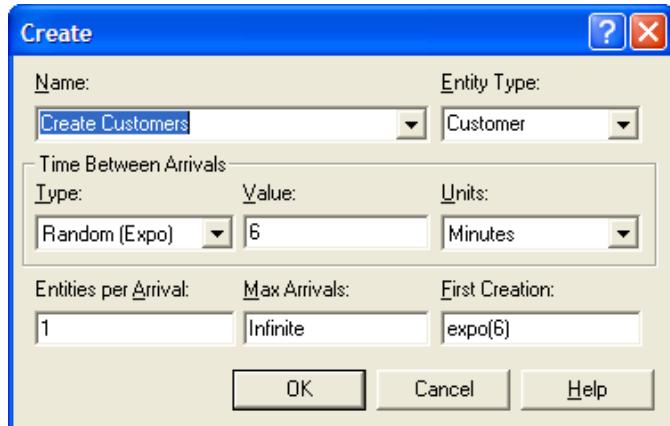


Figure 2.31.: Pharmacy model CREATE module

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 2.32.: Pharmacy model ENTITY module

2.6.5. Specifying 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 data sheet window (Figure 2.33) should have changed to reflect this selection. Double-click on the row in the spreadsheet module for the RESOURCE module to add a resource to the model.

Resource - Basic Process						
	Name	Type	Busy / Hour	Idle / Hour	Per Use	Report Statistics
Double-click here to add a new row.						

Figure 2.33.: RESOURCE module in data sheet view

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 2.34 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 one. Now, you have to indicate how the customers will use this resource within a process.

2. Introduction to Simulation and Arena

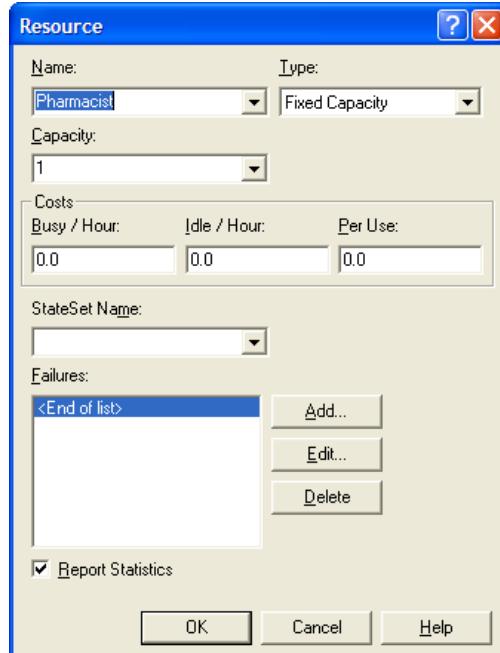


Figure 2.34.: RESOURCE dialog box

2.6.6. 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 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 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 the following pseudo-code.

```
CREATE customers with EXP0(6) time between arrivals
SEIZE 1 pharmacist
DELAY for EXP0(3) minutes
RELEASE 1 pharmacist
DISPOSE customer
```

Open up the PROCESS module and fill it out as indicated in Figure 1.23. 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 by the customer. Within the “Delay Type” area, choose Expression as the type of delay and type in `expo(3)` as the expression.

2.6. Modeling a Simple Discrete-Event Dynamic System

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. When a PROCESS module uses the (Seize, Delay, Release) option, Arena automatically translates this to individual SEIZE, DELAY, and RELEASE modules, just like outlined in the pseudo-code. These modules (SEIZE, DELAY, RELEASE) are executed individually as the entities experience the process. It is very useful to understand what happens when these modules are executed.

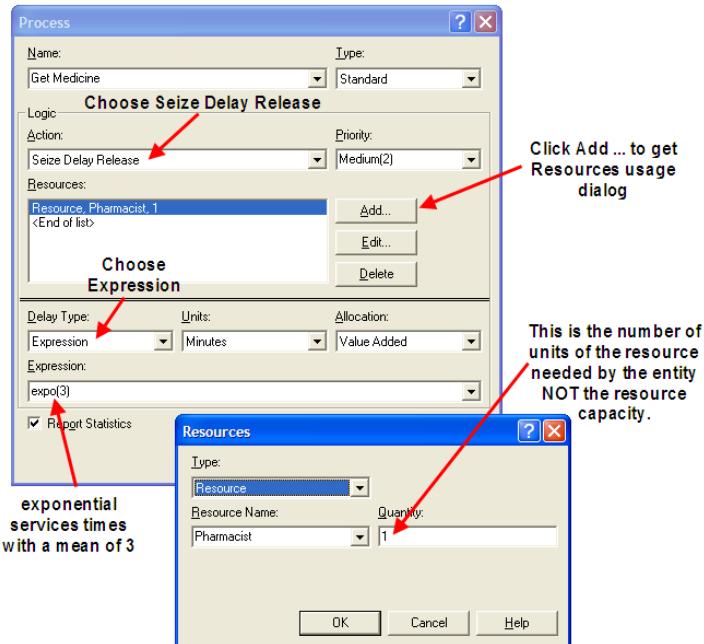


Figure 2.35.: PROCESS module seize, delay, release option

When an entity seizes a resource, the number of units available of the resource's capacity is (automatically) checked against the amount of units requested by the entity. There is a global variable (function) defined for each resource called, $NR(Resource\ ID)$, that returns the number of units of the resource that have been allocated to entity requests. For example, suppose a resource called, $rBankTeller$, has *capacity*, 3, and one of the 3 units (tellers) has already been allocated to a customer for use. That is, one of the tellers is currently serving a customer. In this case, $NR(rBankTeller)$, will have the value 1. There is also a global variable (function) called $MR(Resource\ ID)$ that holds the number of units of capacity defined for the resource. In this example, $MR(rBankTeller)$ has the value 3. In the drive through pharmacy example the value of $MR()$ is 1. Again, $MR()$ holds the current *capacity* of the resource. Obviously, resources can have capacity value of various amounts.

The number of available units of the resource is always the difference, $MR(Resource\ ID) - NR(Resource\ ID)$. In this bank example, the value of $MR(rBankTeller) - NR(rBankTeller)$ is currently 2 ($3 - 1 = 2$) because a customer is in service. Thus, an arriving customer that wants

2. Introduction to Simulation and Arena

one teller will not have to wait in the queue associated with the SEIZE module and will be immediately allocated one of the available tellers. The value of $NR(rBankTeller)$ will be incremented by 1 unit and the number of available units, $MR(rBankTeller) - NR(rBankTeller)$, will decrease by 1. If an entity requests more than the current number of available units of the resource, the entity will be *automatically* placed in the queue associated with the SEIZE module.

The units of a resource that have been allocated to an entity by a successful SEIZE will remain allocated to that entity until those units are released via a RELEASE module. An entity can release 1 or more units of the allocated resource when executing a RELEASE module. When an entity executes a RELEASE module the allocated units of the resource are given back to the resource and $NR(Resource\ ID)$ is *decremented* by the amount released. When the units of the resource are released, the queue associated with the seize (with the highest seize priority) is checked to see if any entities are waiting and if so, those entities are processed to check if their number of units requested of the resource can be allocated. The processing of the queue is based on the queue processing rule (e.g. FIFO (first in, first out), LIFO (last in, first out), random, and priority ranking).

Suppose that a resource has capacity of 1 unit. If there are 2 processes (or seizes) that seize the same resource and entities are waiting in separate queues, the entity that activated the seize with the highest priority will get the resource first. If both seize requests have the same priority, then the entity that activated its seize first will go first (first come first served). The priority field for the seize controls the priority of the seize. It can be anything that evaluates to a number. A resource is considered busy if *at least 1 unit* of its capacity has been allocated. A resource is considered idle when *all* units are idle and the resource is not failed or inactive. The failed and inactive states of resources are discussed in Section 6.2.



A common misunderstanding by novice modelers is to think that they must continually check $MR(Resource\ ID) - NR(Resource\ ID)$ to see if the resource is available. Do not think this way. The purpose of the SEIZE and RELEASE constructs is to manage the allocation and deallocation of resource units. While there can be legitimate reasons for using the values of $NR(Resource\ ID)$ and $MR(Resource\ ID)$ within a model, you should not make the mistake of polling resources for availability. The resource allocation process is done automatically using SEIZE and RELEASE. If you do not like how the resource units are being allocated, then you should use more advanced concepts as discussed in Chapter 6.

Now we are ready to specify how to execute the pharmacy model.

2.6.7. Specify Run Parameters

Let's 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

2.6. Modeling a Simple Discrete-Event Dynamic System

utilization of the pharmacist. This kind of simulation is called an infinite horizon simulation and will be discussed in more detail in Chapter 5.

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 don't want to wait that long! For the sake of simplicity, assume that 10,000 hours of operation is long enough. Within the 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 2.36. 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.

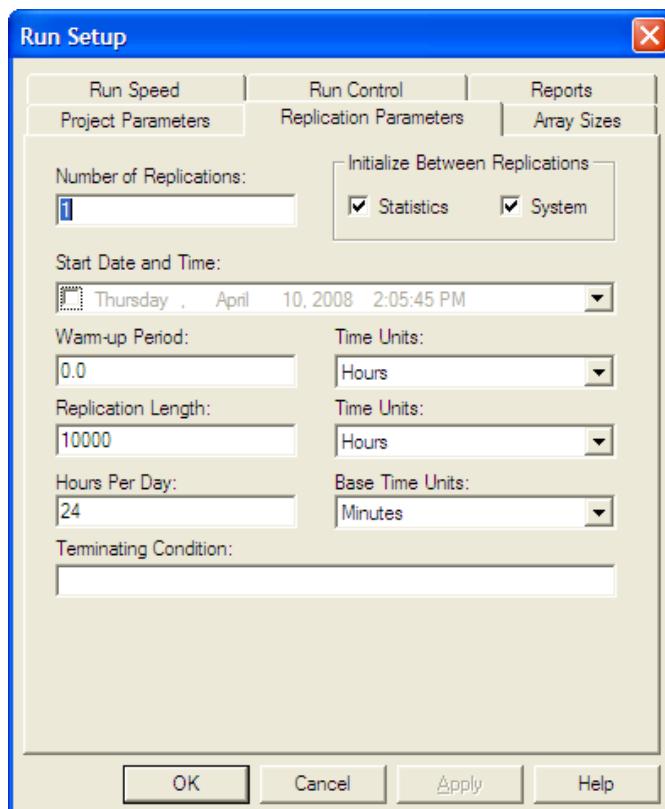


Figure 2.36.: Run setup replication parameters tab

The fully completed model is available within this chapter's files as file, *PharmacyModel.doe*.

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 2.37). 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

2. Introduction to Simulation and Arena

the simulation to stop. The animation slider causes the animation to slow down (to the left) or speed up (to the right).

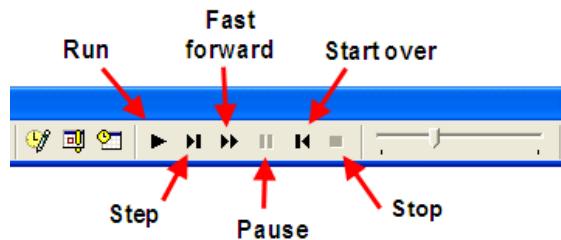


Figure 2.37.: Run toolbar

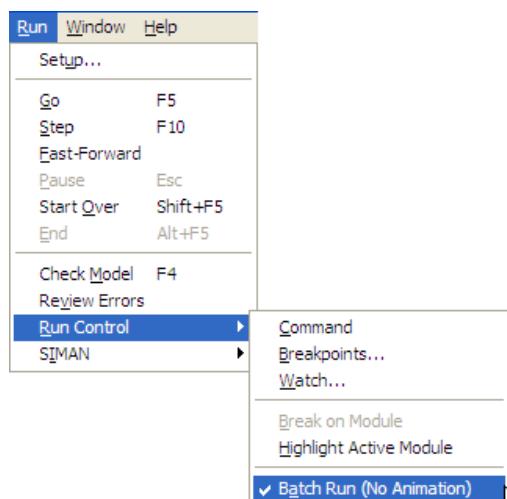


Figure 2.38.: Batch run no animation option

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 2.38 and then when you use the VCR Run button the model will run much faster without animation.

2.6.8. Analyze the Results

After running the model, you will see a dialog (Figure 2.39) indicating that the simulation run is completed and that the simulation results are ready. Answer yes to open up the report viewer.

2.6. Modeling a Simple Discrete-Event Dynamic System

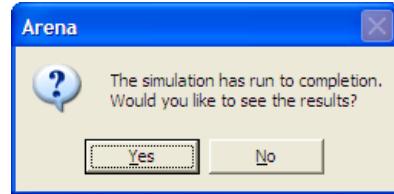


Figure 2.39.: Run completion dialog

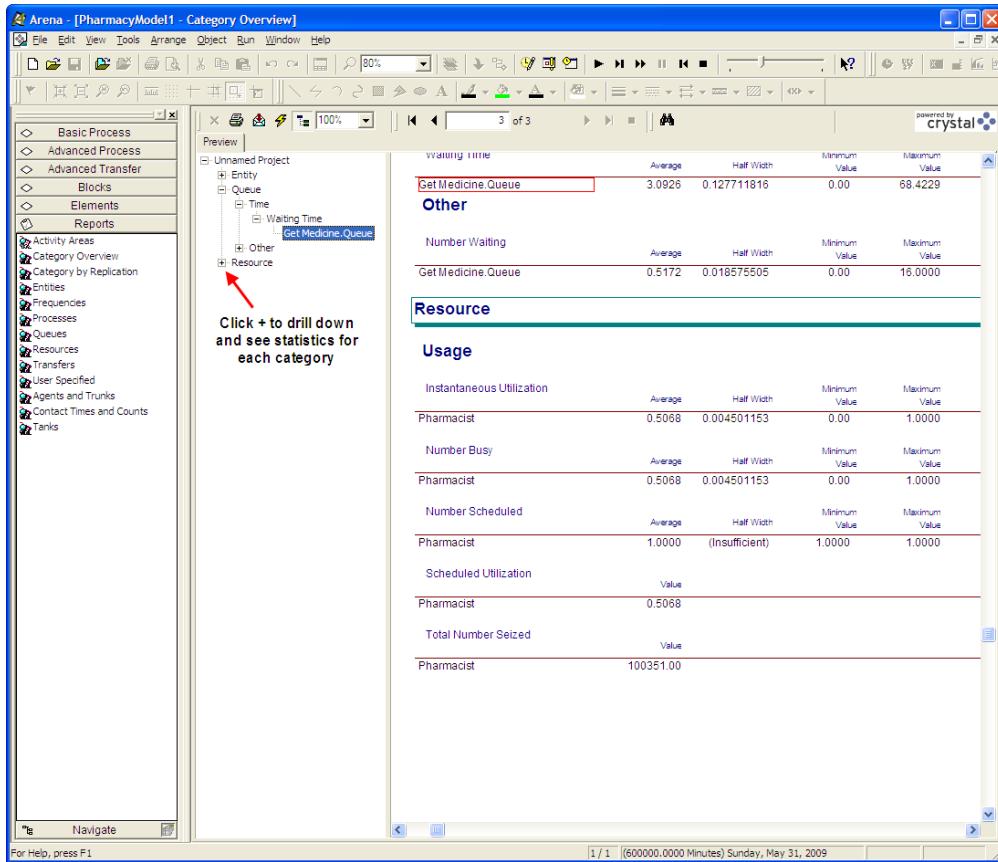


Figure 2.40.: Results for pharmacy model

2. Introduction to Simulation and Arena

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 2.40. The reports indicate that customers wait about 3 minutes on average in the line. reports the average of the waiting times in the queue as well as the associated 95% confidence interval half-width. In addition, reports the minimum and maximum of the 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 average waiting for service. Using the Reports panel in the Project Bar you can also select specific reports.

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 1.30. This file can be read with any text editor such as Notepad, see Figure 2.41.

```

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 2.41.: Text file summary results

Also as indicated in Figure 2.42, Arena generates additional files when you run the model. The *modelname.accdb* file is a Microsoft Access database that holds the information displayed by the report writer. The *modelname.p* file is generated when the model is checked or run. If you have

2.6. Modeling a Simple Discrete-Event Dynamic System

errors or warnings when you check your model, the error file will also show up in the directory of your *modelname.doe* file

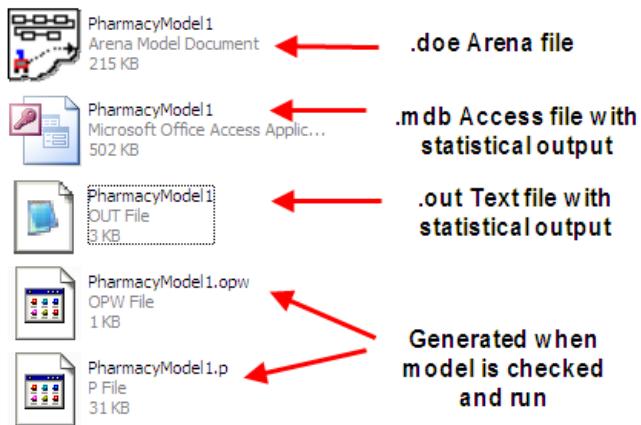


Figure 2.42.: Files generated when running Arena

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 Appendix C 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:

$$L_q = \frac{\rho^2}{1 - \rho}$$

$$\rho = \lambda/\mu$$

$$\lambda = \text{arrival rate to queue}$$

$$\mu = \text{service rate}$$
(2.7)

In addition, the expected waiting time in queue is given by $W_q = L_q/\lambda$. In the pharmacy model, $\lambda = 1/6$, i.e. 1 customer every 6 minutes on average, and $\mu = 1/3$, i.e. 1 customer every 3 minutes on average. The quantity, ρ , is called the utilization of the server. Using these values in the formulas for L_q and W_q results in:

2. Introduction to Simulation and Arena

$$\begin{aligned}\rho &= 0.5 \\ L_q &= \frac{0.5 \times 0.5}{1 - 0.5} = 0.5 \\ W_q &= \frac{0.5}{1/6} = 3 \text{ minutes}\end{aligned}$$

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, e.g. lognormal. 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 3 or more cars waiting.

2.7. 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's assume that if there are four or more customers in the line when a customer arrives that 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 2.43 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 upon 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 called `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 following pseudo-code illustrates these ideas.

```
CREATE customers with EXPO(6) time between arrivals
DECIDE IF NQ(Get Medicine.Queue) <= 3 customers
SEIZE 1 pharmacist
```

2.7. Extending the Drive Through Pharmacy Model

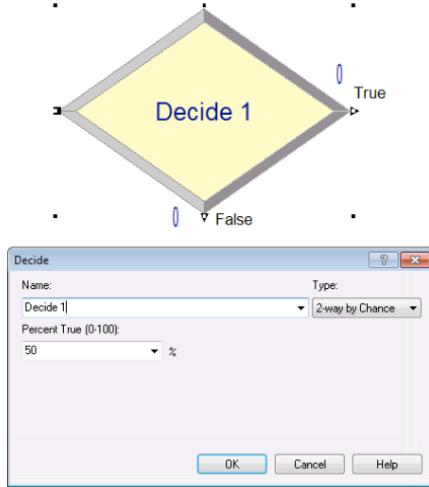


Figure 2.43.: DECIDE flowchart symbol and module dialog

```

DELAY for EXPO(3) minutes
RELEASE 1 pharmacist
DISPOSE customer

```

Based on this pseudo-code, the customer only enters the PROCESS module if there are 3 or less cars waiting in line. The overall model is illustrated in Figure 2.44. Notice that there are now two DISPOSE modules, one for customers who do not enter (balk) and one for those that enter and complete service. Figure 2.45 shows the use of the NQ() function to chose the correct path.

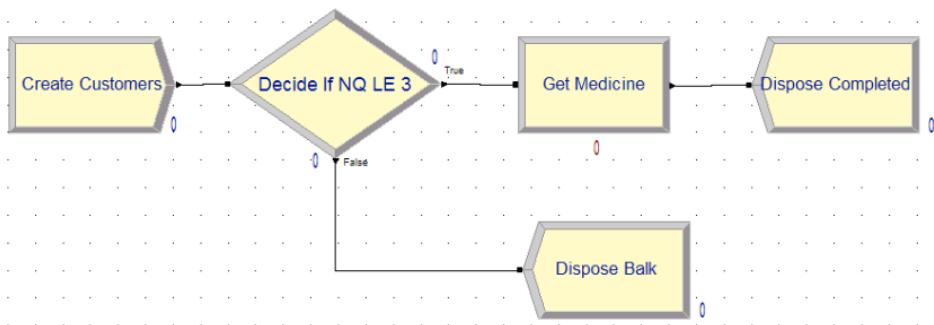


Figure 2.44.: Overview of pharmacy model with DECIDE module

Running the model using the same conditions as before results less waiting and utilization as show in Figure 2.46. 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.

2. Introduction to Simulation and Arena

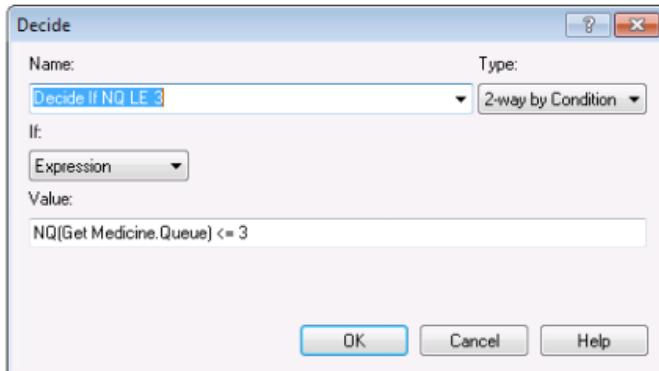


Figure 2.45.: DECIDE module dialog with condition specified

Time		
Waiting Time	Average	Half Width
Get Medicine.Queue	2.5724	0.077521344
Other		
Number Waiting	Average	Half Width
Get Medicine.Queue	0.4213	0.011593416
Resource		
Usage		
Instantaneous Utilization	Average	Half Width
Pharmacist	0.4943	0.004910555
Number Busy	Average	Half Width
Pharmacist	0.4943	0.004910555

Figure 2.46.: Results for pharmacy model with DECIDE module

2.8. 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. There are 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 will illustrate 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. There are 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 2.47. You can select different picture files by navigating to where they are installed within your Arena installation. See Figure 2.48. This may vary by installation. For example, the picture library can be found at the following location in my installation `C:\Users\Public\Documents\Rockwell Software\Arena\PictureLibraries`.

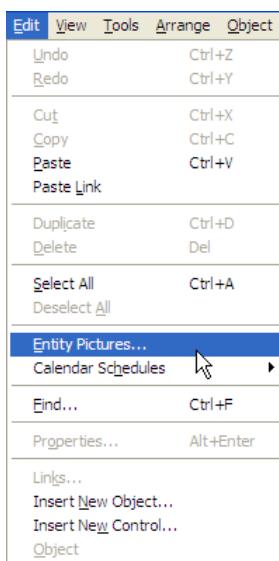


Figure 2.47.: Accessing entity pictures

Once you have selected Edit > Entity Pictures, you should see the entity picture placement dialog as shown in Figure 2.49. 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 picture libraries within the distribution folder on your hard-drive (see Figure 2.48) you can select the `Vehicles.plb` file. When selected as the current library, your entity picture placement dialog should look as shown in Figure 2.49.

Scroll down in the vehicle library and 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.

2. Introduction to Simulation and Arena

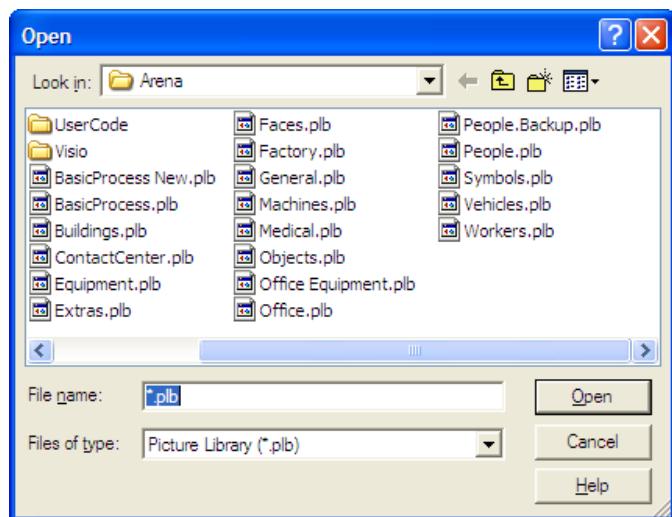


Figure 2.48.: Entity pictures folder

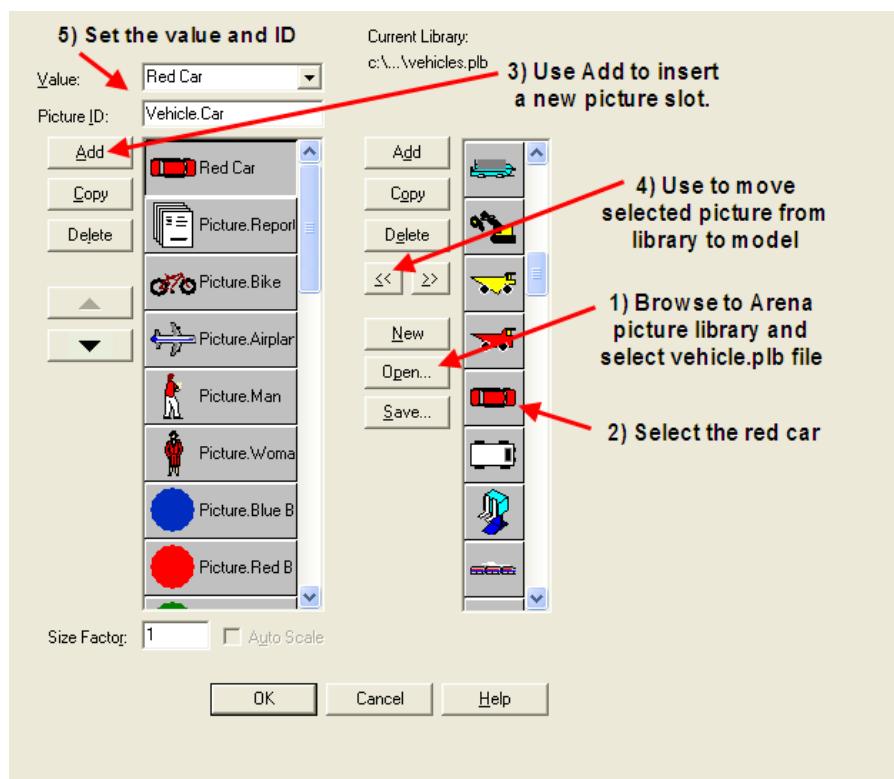


Figure 2.49.: Entity picture placement dialog

2.8. Animating the Drive Through Pharmacy Model

Now, you will use Arena's drawing tools to draw the outline of the pharmacy building. has standard drawing tools (e.g. lines, rectangles, polygons, etc) as well as way to add text, fill options etc. These tools are available through the Drawing toolbar as shown in Figure 2.50. 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.

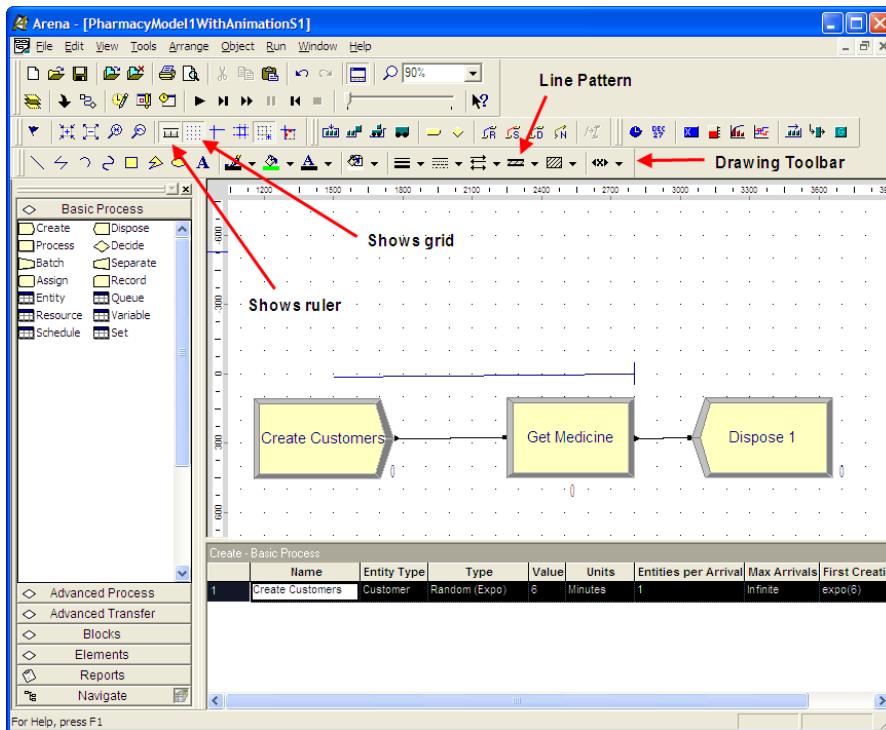


Figure 2.50.: Basic drawing toolbar

Figure 2.51 illustrates the background drawing for the pharmacy. The drawing editor works like most computer based drawing editors. You should just drag and drop the items, set the fill options, thickness, and line pattern as you go. This portion of this example is available in the *PharmacyModel1WithAnimationS1.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 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 1 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.

2. Introduction to Simulation and Arena

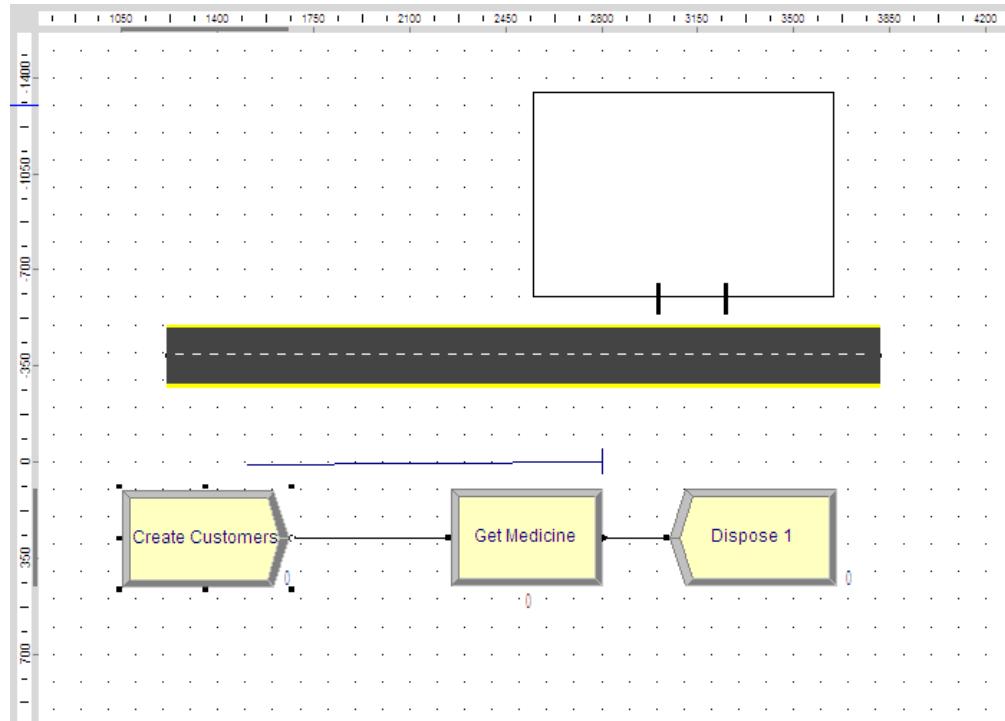


Figure 2.51.: Simple pharmacy drawing within Arena model Window

In the case of the pharmacy model, since there is only 1 resource unit (the pharmacist), the resource is busy when the sole pharmacist is 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.
3. Select the over head person view picture from the library's list of pictures, see Figure 2.52
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 un-group 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 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.) 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 that 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 2.53. Once you press OK, the cursor will turn in to a cross-hair and allow you to place the animation queue.

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 2.54. The cursor should turn to cross-hairs and you should place the variable animation inside the pharmacy as shown in Figure 2.55.

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 2.56.

The final version of the animated pharmacy model is available in the file, *PharmacyModuleWithAnimationS2.doe*. If you run the model with the animation on, you will see the 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.

2. Introduction to Simulation and Arena

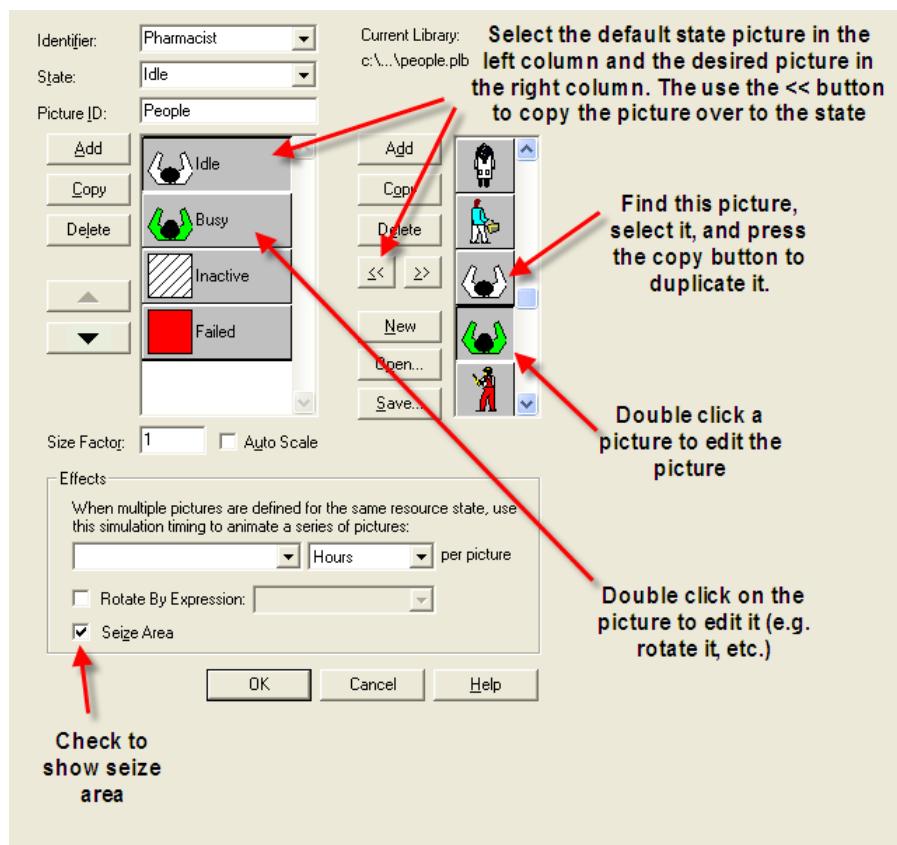


Figure 2.52.: Resource animation state picture assignment

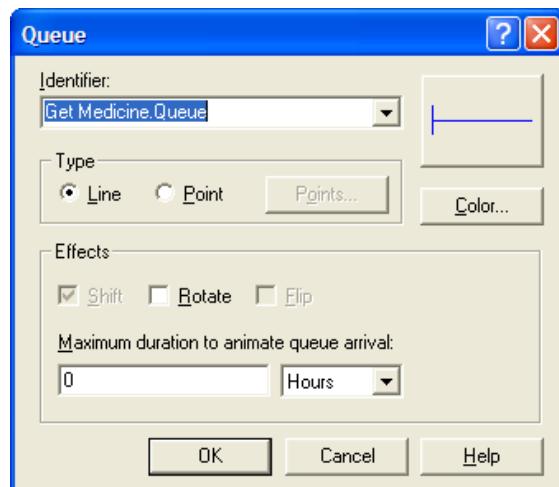


Figure 2.53.: Queue animation dialog

2.8. Animating the Drive Through Pharmacy Model

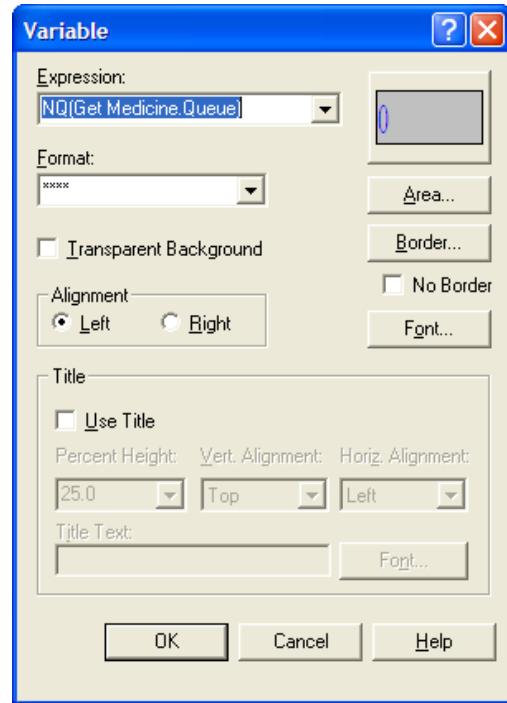


Figure 2.54.: Variable animation dialog

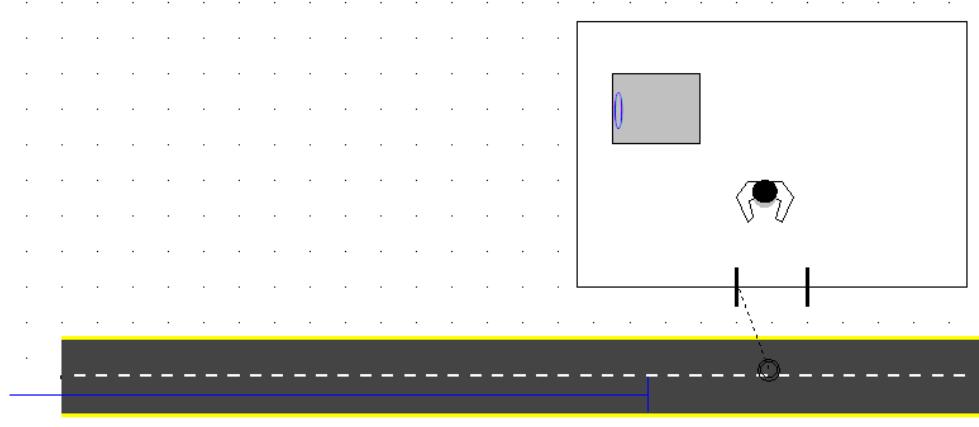


Figure 2.55.: Final animation for pharmacy example

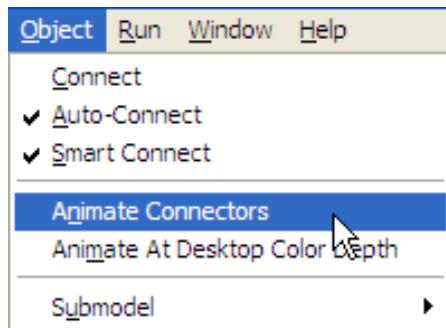


Figure 2.56.: Disabling connector animation



How can I animate a resource that has capacity > 1 This question is commonly asked because you want to show each unit of the resource being busy. Unfortunately, this isn't easy to do because a resource is considered busy if at least 1 unit of its capacity is busy. You could animate the `NR(resource_name)` variable, see Section 6.2, to show how many are busy (either numerically or with a level animation). Also, by showing the seize point and adding multiple points you can show each entity as it is using the resource. There is also something called a storage, see Section 4.1.4, which will do just about the same thing. You can also animate 1-D variables by having the variable indicate whether each unit is busy or not. All of these methods are illustrated in the attached Arena model. This should approximate what you are trying to do. The variable method is probably the best but takes the most work. If you are going to use the variable method, then you might just look at using a resource set instead. If you want to use a resource animation and show each separately, then you need to define a different resource to represent each unit of the overall resource, put them in a set and seize/delay/release from the set. Then, you can animate each of the individual resources in the standard manner. Please see the Arena file, *ResourceAnimations.doe* that accompanies this chapter's files for an example.

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 a 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 which can be easily accessed via the Navigate bar. Many detailed examples of what is possible with animation come as demo models with the distribution. For example, you should explore the models in the Examples folder (e.g. *FlexibleManufacturing.doe*). If you find that the drawing tools are limiting, then you can use other drawing programs to make the drawings and import them as background for your model. 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 7 returns to more on the topic of animation, when material handling constructs (e.g. conveyors and transporters) are examined.

2.9. Attributes, Variables, and Some I/O

In this section, we will dig deeper into the concepts of attributes and variables. In addition, an introduction to some additional statistical collection will be discussed. Finally, we will see how to write data to a file. To illustrate these concepts, we will expand upon the pharmacy model.

2.9.1. Modifying the Pharmacy Model

Suppose that customers who arrive to the pharmacy have either 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 2.17.

Table 2.17.: Prescription related data

# Prescription	Chance	Service Time Mean
1	50%	3 minutes
2	30%	4 minutes
3	20%	5 minutes

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, and the time of arrival, and the total time spent in the system

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.

2. Introduction to Simulation and Arena

As we saw earlier in the chapter, you should follow a basic modeling recipe and address the following steps:

- *What is the system?*
 - *What are the elements of the system?*
 - *What information is known by the system?*
- *What are the required performance measures?*
- *What are the entity types?*
 - *What information must be recorded or remembered for each entity instance?*
 - *How are entities (entity instances) introduced into the system?*
- *What are the resources that are used by the entity types?*
 - *Which entity types 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*

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 inter-arrival 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 shown in Figure 2.29 because the activity and use of resources of the customer has not changed.

When implementing this model, 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.
- ATTRIBUTE This data module will be used to define attributes associated with the customer.

Thus, the new modules are the READ/WRITE module, the FILE module, and the RECORD module.

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 1-D variable (`vNP`) with 3 elements can be used to track the number of customers requiring each of the 3 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 a specific customer instance was stored in a (global) variable, then during the movement cycle of the other entities (customers), some other entity instance might change the value of the variable. Thus, the information concerning the current customer's number of prescriptions would be over written by the next customer that enters the system. Thus, this information must be carried with the entity and not stored globally. Global variables are very useful for communicating information between entities; however, as in any programming language care must be taken to use them correctly.

2. Introduction to Simulation and Arena



What is the difference between attributes and variables? The key difference is that a variable is global. A variable belongs to the system. Attributes are attached to an entity instance. Variables define a place in memory in which data can be stored. Since the memory is global to the model, any module or entity can read/write/use the value of the variable. Memory allocated to an attribute is stored in the *entity table* with the specific entity instance as discussed in Section 2.5.1. If you have experience with a programming language like Java, then Arena variables are like static class variable, which is shared by all instances of the class. Arena attributes are like Java fields of the class, which are attached only to the object instances.

The information in Table 2.17 also needs to be modeled. Does this information belong to the system or to each customer? While the information is *about* customers (in general), each customer does not need to "carry" this information. The information is really about how to create the customers. In other words, it is about the entity type not the entity instance. 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 1-D variable with 3 elements representing the mean of each distribution. The basic pseudo-code for this situation is given in the following pseudo-code.

```
CREATE customers with EXPO(6) time between arrivals
BEGIN ASSIGN
    myNP = DISC(0.5, 1, 0.8, 2, 1.0, 3)
    vNumPrescriptions = vNumPrescriptions + myNP
    vNP(myNP) = vNP(myNP) + 1
    myArriveTime = TNOW //TNOW represents the current simulation time
END ASSIGN
WRITE customer arrival information to a file
BEGIN PROCESS
    SEIZE 1 pharmacist
    DELAY for EXPO(3) minutes
    RELEASE 1 pharmacist
END PROCESS
RECORD (TNOW - myArriveTime)
WRITE customer departure information to a file
DISPOSE customer
```

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 Section 2.6.2. You should start by defining the variables for the example. With your copy of the model open, go to the Basic Panel and 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 1-D variable should have 3 rows and should count the number of customers having 1, 2, 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 1-D variable should have 3 rows representing the mean of the service distributions for the 3 prescription quantities. The index into the array is the prescription quantity. The elements (1, 2, 3) of this variable should be initialized at the values 3, 4, 5 respectively.

Given the information concerning the variables, the variables can be defined as shown in Figure 2.57. For scalar variables, you can tell Arena to automatically report statistics on the time average value of the variable. Chapter 3 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.

	Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics
1	<code>vNumPrescriptions</code>			Real	System	0 rows	<input checked="" type="checkbox"/>
2	<code>vNP</code>	3		Real	System	0 rows	<input type="checkbox"/>
3	<code>vServiceMean</code>	3		Real	System	3 rows	<input type="checkbox"/>
4	<code>vEventType</code>			Real	System	0 rows	<input type="checkbox"/>

Double-click here to add a new row.

Initial Values	
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 2.57.: Defining the variables

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.

2.9.2. 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 sim-

2. Introduction to Simulation and Arena

ply represents a series of logical assignment, e.g. Let $X = 2$, as you would have in any common programming language. Figure 2.58 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 `DISC(0.5, 1, 0.8, 2, 1.0, 3)`.

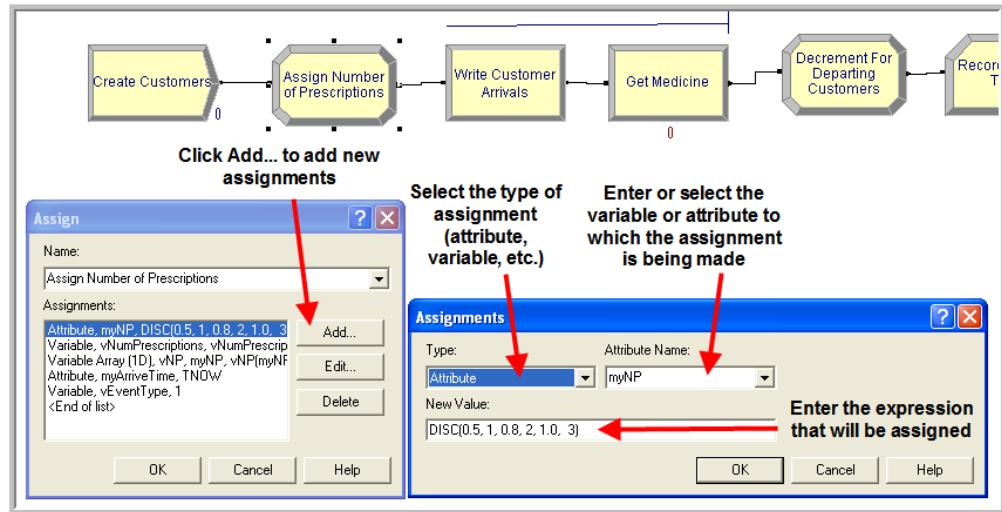


Figure 2.58.: Assigning the number of prescriptions

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, i.e. $P(X = v_i) = cp_{i+1} - cp_i$ 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 2.59 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 2.60.

You should complete your ASSIGN module as shown in Figure 2.60. The data sheet view of Figure 2.60 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

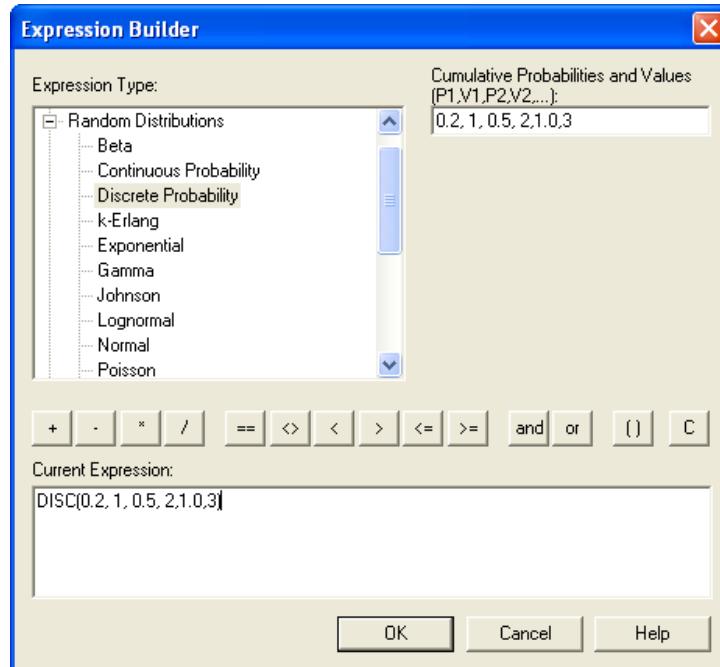


Figure 2.59.: Using the expression builder

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 2.60.: Data sheet view of ASSIGN module

2. Introduction to Simulation and Arena

customers in the system with the given number of prescriptions. The fourth assignment uses the variable `TNOW` to assign the 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.

2.9.3. Using the READWRITE Module

As with any programming language, has the ability to read and write information from files. Often the READWRITE module is left to the coverage of miscellaneous topics; 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 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 , but this example uses the READWRITE module so that you can gain an understanding of how 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 2.61. 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 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. 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 2.62. 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:

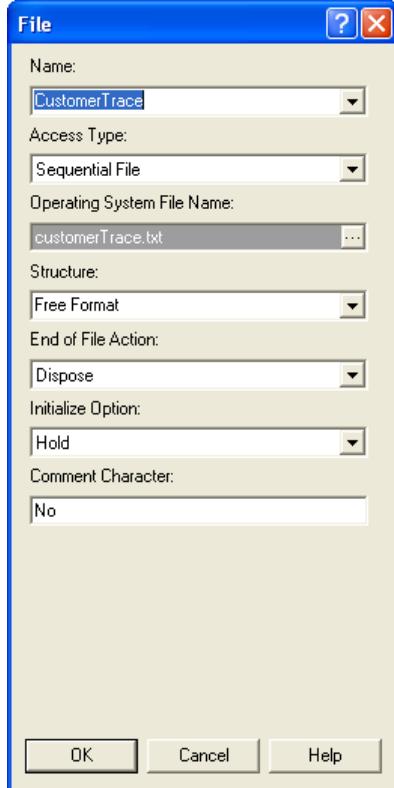


Figure 2.61.: The FILE module

- `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 a simulation.

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 2.63, we use the 1-D 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

2. Introduction to Simulation and Arena

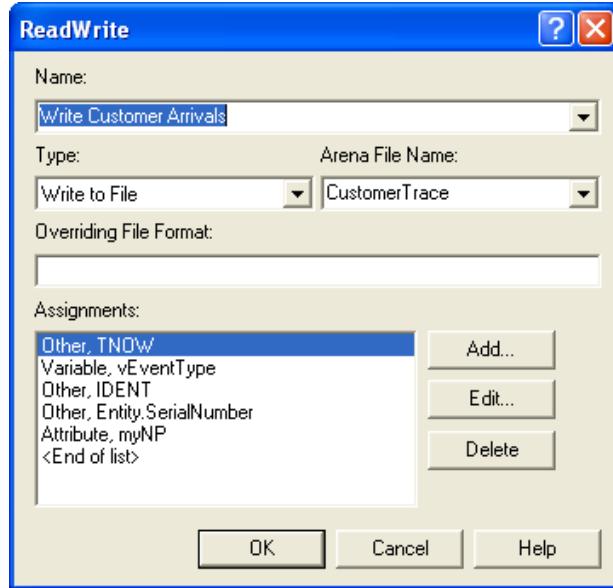


Figure 2.62.: The READWRITE module

prescriptions for the customer. This value is used as the index into the `vServiceMean` 1-D variable to select the appropriate mean. Since has limited data structures, the ability to use arrays in this manner is quite common.

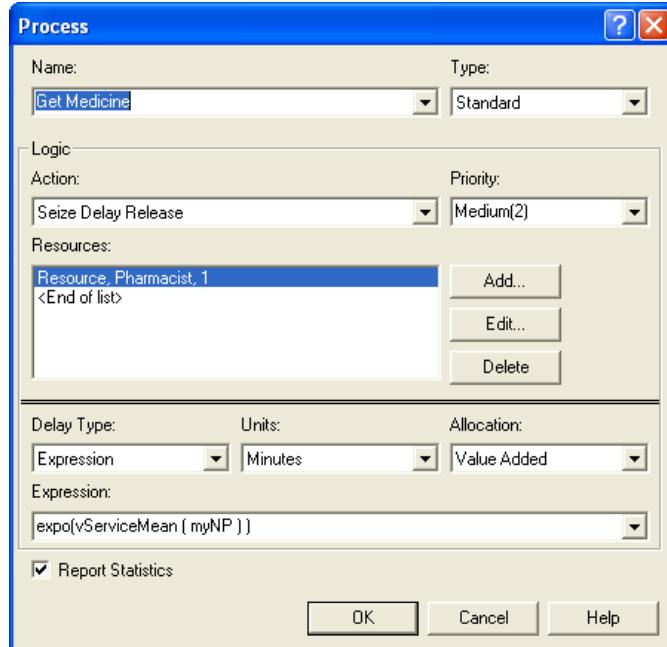


Figure 2.63.: Changed PROCESS module

Now, that 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 2.64 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.

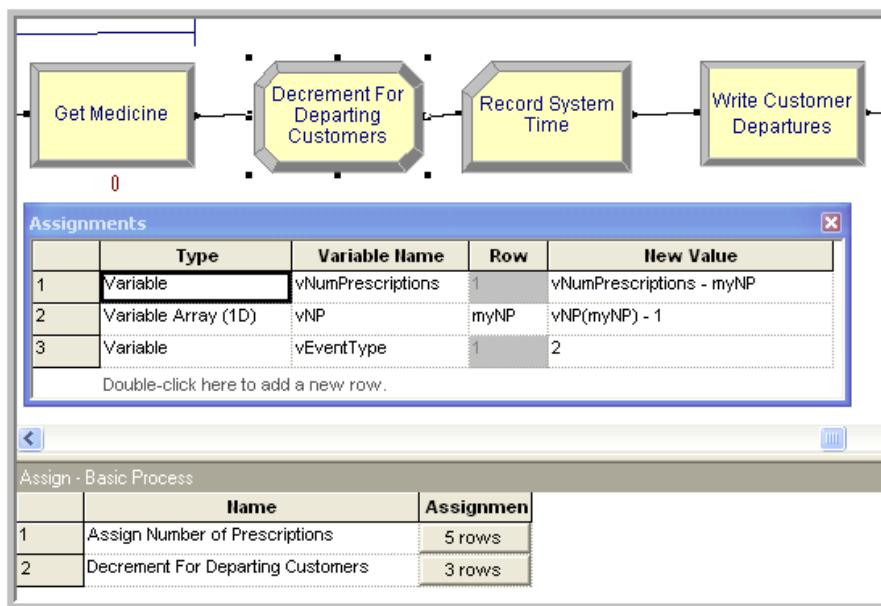


Figure 2.64.: Updating variables after service

2.9.4. 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:

- *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.

2. Introduction to Simulation and Arena

- *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 2.65. 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 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 reports labeled with the name of the tally, e.g. *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.

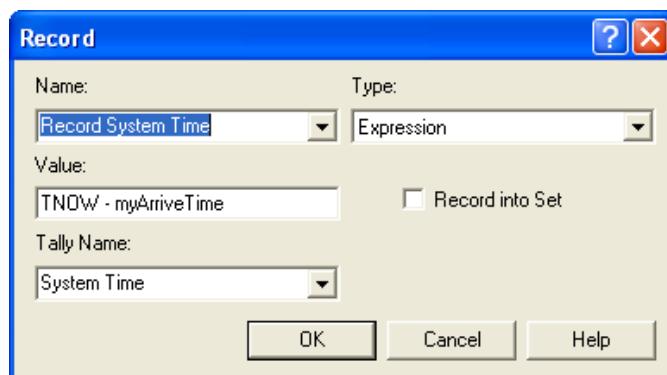


Figure 2.65.: Using the RECORD module to capture system time

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 2.66 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.

2.9.5. 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 2.67 shows the animation toolbar within . If the toolbar is not showing in your Environment, then you can right-click in the toolbar area and make sure that the Animate option is checked as shown in Figure 2.67.

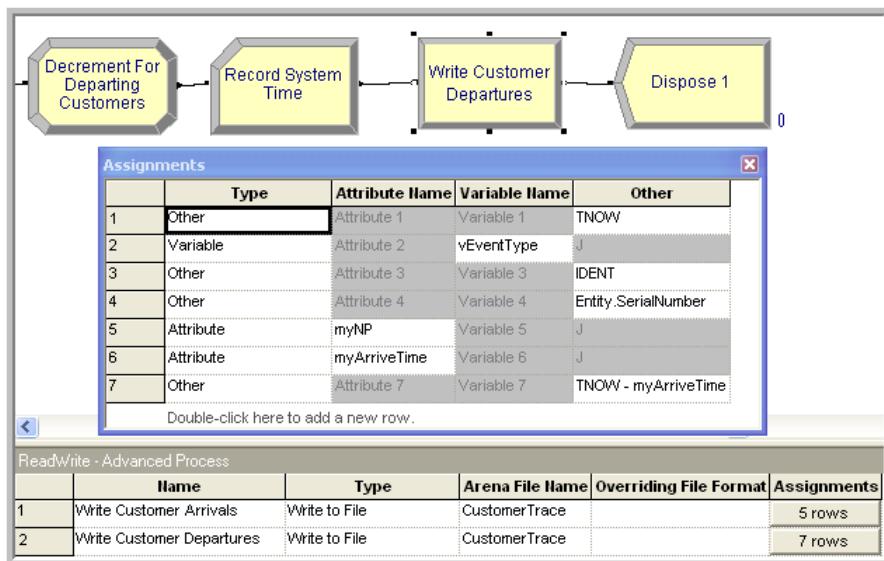


Figure 2.66.: READWRITE module for departing customers

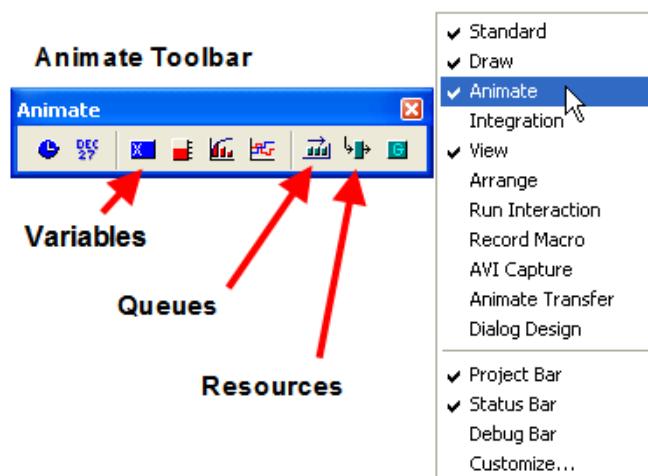


Figure 2.67.: The Animate toolbar

2. Introduction to Simulation and Arena

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 2.68. 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 2.69. During the simulation run, this area will display the variable vNumPrescriptions as it changes with respect to time.

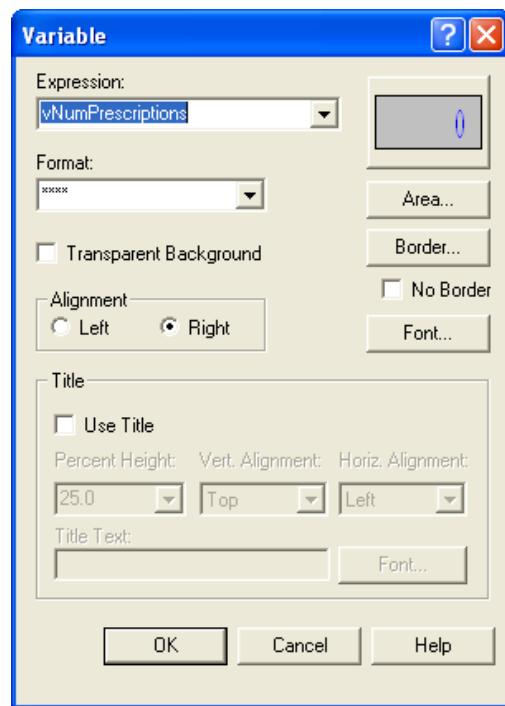
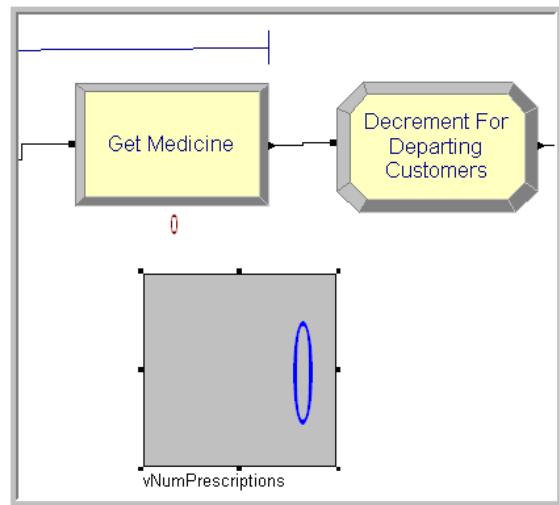
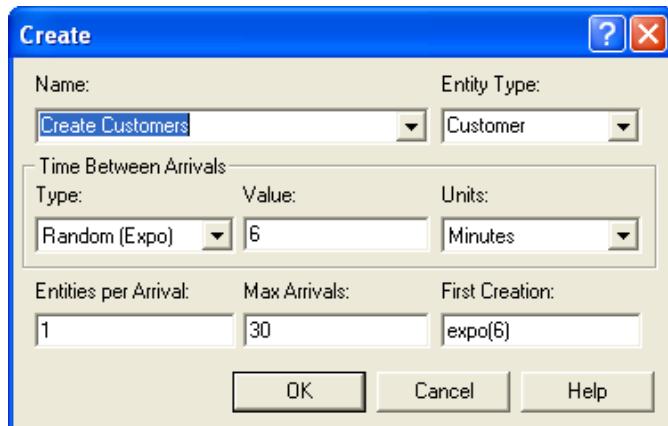


Figure 2.68.: Animate variable dialog

2.9.6. 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 Figure 2.70 and Figure 2.71) ensure that the simulation will stop after 30 customers are created. If you do not specify a replication length for the simulation, it will process whatever entities are created within the model until there are no more entities to

**Figure 2.69.: Animation of the prescriptions****Figure 2.70.: Creating only 30 customers**

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 file, *Pharmacy-ModelRevised.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.

2. Introduction to Simulation and Arena

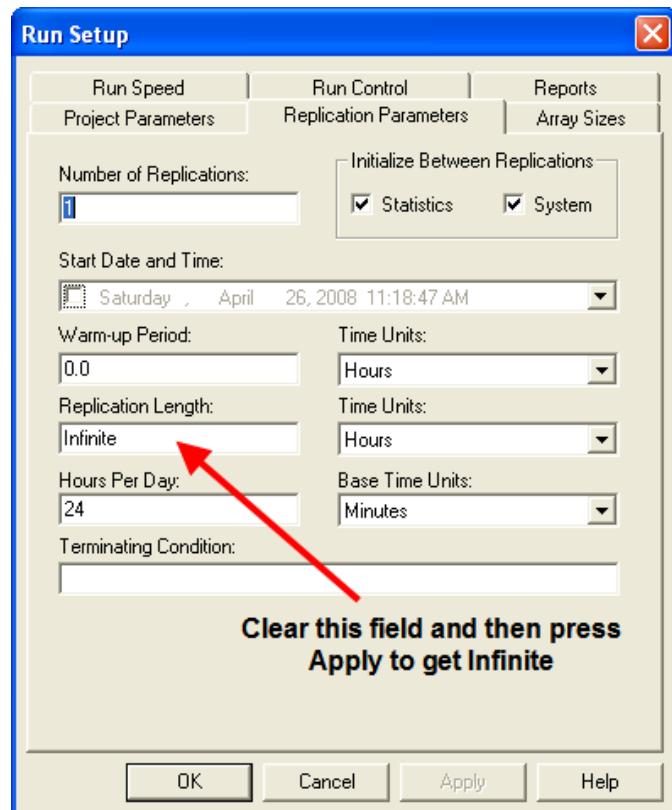


Figure 2.71.: Run setup with no replication length

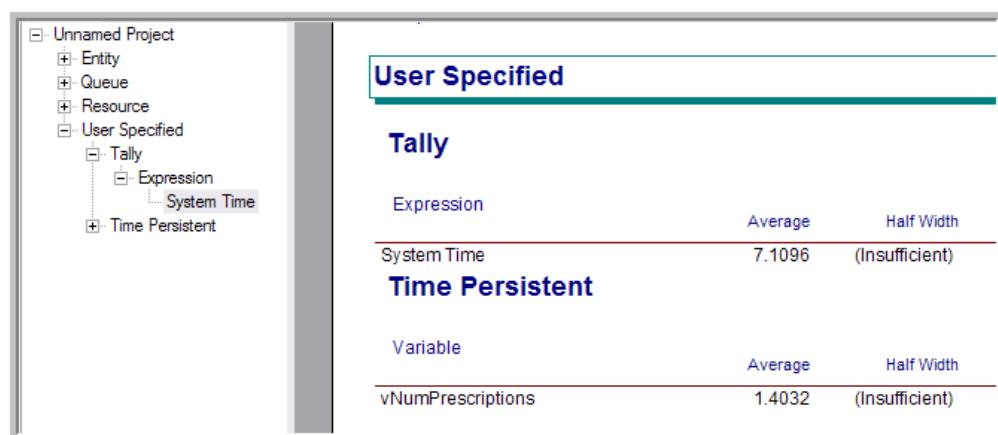


Figure 2.72.: System time output report

Figure 2.72 indicates that the average system time of the 30 customers that 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 be a text file called *customerTrace.txt* in the same directory as where you ran the model. Table 2.18 shows the data from the first 5 customers through the system.

Table 2.18.: 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 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 2nd customer, (IDENT = 3, serial number 2) departed at time 13.31427. Thus, the 3rd and 4th customers had to wait in 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 shown on the output 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 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. Future chapters will illustrate these more complex flow processes. In the next section, we will study how Arena processes entities and events. This will provide a deeper understanding of what is going on "under the hood".

2.10. How Arena Manages Entities and Events

This section will provide 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). To get a more detailed step by step view of Arena's operation, I recommend using the Arena Run Controller as noted in Appendix D.2. The Arena Run Controller facilitates the tracking and tracing of entities as they move through the system. This section should help build a solid conceptual understanding for how entities are processed.

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 to occur that change the state of the system. 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, 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 clock update phase 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 entity movement phase. Thus, a key to understanding 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 but 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) divide 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, i.e. 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 entity movement phase 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 where the an entity is not waiting on time or a condition that can be automatically processed within the model. The dormant state represents the situation where 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 2.73 illustrates how an entity can transition between the five states, based on Figure 24.9 of (Schriber and Brunner, 1998). 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 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.

2. Introduction to Simulation and Arena

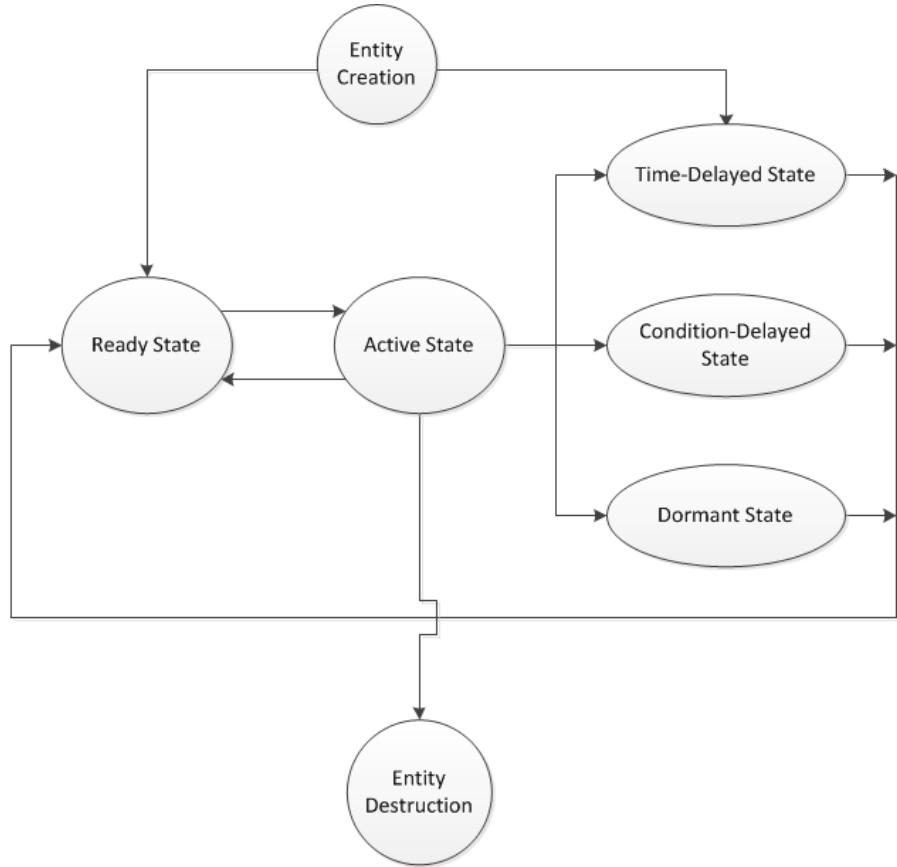


Figure 2.73.: Illustration of entity state transitions

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 current events list holds all the entities that are candidates to start moving at the current time. The list is processed in a first 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 future events list (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 delay lists within the model at any time. For example, the BATCH module's queue is also a delay list.

Finally, entities that are in the dormant state are held in user managed lists (UML). In these are queues that are not tied to a specific condition. HOLD modules allow for an “infinite hold” option, which defines a user managed list. 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 2.19 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.

Table 2.19.: Entities conceptualized as records

IDENT	Entity.SerialNumber	Size	Weight	ProcessingTime	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

The entity movement phase and clock update phase within 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
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 and it is processed as previously described. When there are no more entities on the CEL and no qualifying condition delayed entities, the clock update phase begins.

2. Introduction to Simulation and Arena

5. The clock update phase 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. 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 entities are processed. 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. (Schriber and Brunner, 1998) discuss 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.

2.11. 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 Arena modules covered included:

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.

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.

Arena has many other facets that have yet to be touched upon. Not only does 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 fit models for the probability distributions. Finally, through its integration with OptQuest, can assist in finding optimal design configurations via simulation.

The next chapter will dive deeper into the statistical aspects associated with simulation experiments.

2.12. Exercises

Exercise 2.1. *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.

Exercise 2.2.

- a. 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.
 - b. An _____ is defined as an instantaneous occurrence that may change the state of the system.
 - c. An _____ describes the properties of entities by data values.
-

Exercise 2.3. 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

Exercise 2.4. The service times for a 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 rate parameter of the exponential distribution is $\lambda = 45$ retrievals per second. Setup an model that will generate 20 observations of the retrieval times. Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.

Exercise 2.5. The time to failure for a computer printer fan has a Weibull distribution with shape parameter $\alpha = 2$ and scale parameter $\beta = 3$. Setup an 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.

Exercise 2.6. 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 model to generate 100 observations from this truncated distribution. Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.

Exercise 2.7. The interest rate for a capital project is unknown. An accountant has estimated that the minimum interest rate will 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 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.

Exercise 2.8. Develop an 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.

Exercise 2.9. 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 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.

Setup an 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?

Exercise 2.10. Suppose that X is a random variable with a $N(\mu = 2, \sigma = 1.5)$ normal distribution. Generate 100 observations of X using an 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 $P(X > 3)$ from your observations. Report a 95% confidence interval for your point estimate.

Exercise 2.11. Samples of 20 parts from a metal grinding process are selected every hour. Typically 2% of the parts need 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. Simulate 30 hours of the process, i.e. 30 samples of size 20, and estimate the chance that X exceeds its expected value by more than 3 standard deviations.

Exercise 2.12. Consider the following discrete distribution of the random variable X whose probability mass function is $p(x)$. Setup an 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.

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

Exercise 2.13. The demand for parts at a repair bench per day can be described by the following discrete probability mass function. Setup an model to generate 30 observations of the demand. Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.

Demand	0	1	2
Probability	0.3	0.2	0.5

Exercise 2.14. Using and the Monte Carlo method estimate the following integral with 95% confidence to within ± 0.01 .

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

Exercise 2.15. Using and the Monte Carlo method estimate the following integral with 99% confidence to within ± 0.01 .

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

Exercise 2.16. Using and the Monte Carlo method estimate the following integral with 99% confidence to within ± 0.01 .

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

Exercise 2.17. 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 to 7 years with the chances given in the following table.

2. Introduction to Simulation and Arena

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 percent 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 ± 10 dollars. Develop a model for this situation.

Exercise 2.18. 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 percent 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 to answer the following questions:

Compare the expected present worth of the two alternatives. Estimate the probability that alternative A has a higher present worth than alternative B.

Determine the number of samples needed to be 95% confidence that you have estimated the $P[PW(A) > PW(B)]$ to within ± 0.10 .

Exercise 2.19. 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 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 model to estimate the probability that the gap D is less than 10.1 millimeters with 95% confidence to within plus or minus 0.01 millimeters.

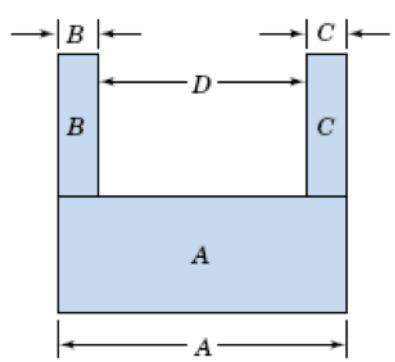


Figure 2.74.: U-Shaped Component

Exercise 2.20. 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.

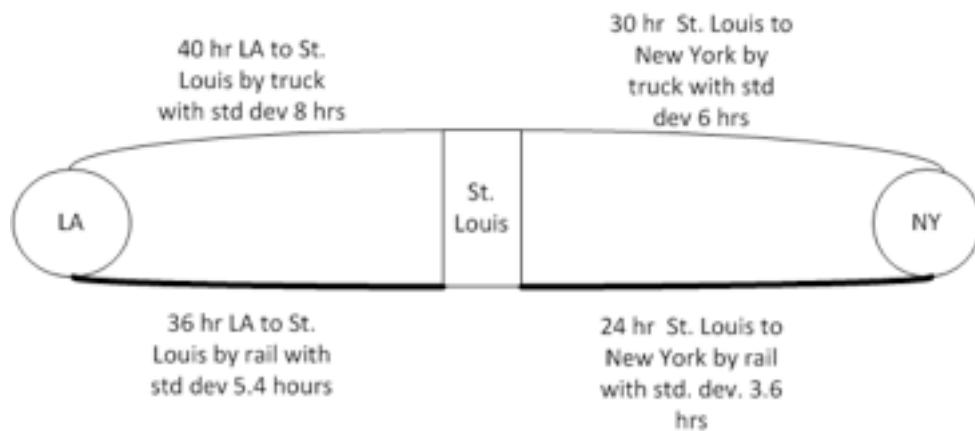


Figure 2.75.: Truck Paths

Assume that the travel times (in either direction) are lognormally distributed as shown in the figure. For example, the time from NY to St. Louis (or St. Louis to NY) by truck is 30 hours with

2. Introduction to Simulation and Arena

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 NY to LA. Develop an simulation for this problem.

- a. How many shipment combinations are there?
 - b. Write an 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.
-

Exercise 2.21. 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	50000	65000	80000	
Probability	0.40	0.30	0.30	

Use a simulation model to generate 1000 observations of the profit.

Estimate the mean profit from your sample and compute a 95% confidence interval for the mean profit. Estimate the probability that the profit will be positive.

Exercise 2.22. 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:

$$\begin{aligned} \text{Profit} = & (\text{revenue from sales}) - (\text{cost of magazines}) \\ & - (\text{lost profit from excess demand}) \\ & + (\text{salvage value from sale of scrap magazines}) \end{aligned}$$

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, 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	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 setup 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 model to estimate the average profit based on 100 observations.

Exercise 2.23. 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

2. Introduction to Simulation and Arena

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 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.

Exercise 2.24. 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 $\text{LDT} = \sum_{i=1}^T D_i$ 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 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.

Assume that the lead-time is a constant 10 days. Develop a 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.

Assume that the lead-time has a shifted geometric distribution with probability parameter equal to 0.2. Use a 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.

Exercise 2.25. If $Z \sim N(0, 1)$, and $Y = \sum_{i=1}^k Z_i^2$ then $Y \sim \chi_k^2$, where χ_k^2 is a chi-squared random variable with k degrees of freedom. Setup a model to generate 50 χ_5^2 random variates.

Report the minimum, maximum, sample average, and 95% confidence interval half-width of the observations.

Exercise 2.26. Setup a model that will generate 30 observations from the following probability density function using the Acceptance-Rejection algorithm for generating random variates.

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

Exercise 2.27. 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, i.e. $N(0, 1)$. Use 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.

Exercise 2.28. 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

Exercise 2.29. 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, $\bar{N}(t)$, 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 .

2. Introduction to Simulation and Arena

t	0	2	4	5	7	10	12	15	20
$N(t)$	0	1	0	1	2	3	2	1	0

Exercise 2.30. Consider the banking situation described within the chapter. 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 re-create 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

Complete a table similar to that used in the chapter and compute the average of the system times for the customers. What percentage of the total time was the teller idle? Compute the percentage of time that there were 0, 1, 2, and 3 customers in the queue.

Exercise 2.31. Consider the following inter-arrival and service times for the first 25 customers to a single server queuing system.

Customer Number	Inter-Arrival 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	

We are given the inter-arrival times. Determine the time of arrival of each customer. Complete the event and state variable change table associated with this situation. 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. 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. 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. Compute the average time spent in the system for the customers.

Exercise 2.32. 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 second. There is no upper limit on the number of parts that wait for process. Setup an model to estimate the expected number of parts waiting in the queue and the utilization of the machine. Run your model for 10000 seconds for 30 replications and report the

2. Introduction to Simulation and Arena

results. Use M/M/1 queueing results from the chapter to verify that your simulation is working as intended.

Exercise 2.33. A large car dealer has a policy of providing cars for its customers that 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. Setup a model to estimate the expected time within the shop for the cars and the utilization of the mechanic. Run your model for 10000 days for 30 replications and report the results. Estimate the total cost per day to the dealer for this policy. Use the M/M/1 queueing results from the chapter to verify that your simulation is working as intended.

Exercise 2.34. 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 the average 82% of the sets pass inspection. The remaining 18% are 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 10000 minutes for 30 replications and report the results.

Exercise 2.35. 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 and discuss the effect on the performance of the drive through lane. Run your model for 1 year, with 20 replications.

Exercise 2.36. The lack of memory property of the exponential distribution states that given Δt 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 Δt . 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 and test the lack of memory property empirically. Explain in your own words what lack of memory means.

Exercise 2.37. 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. Develop a simulation model to estimate the average delay time for a query

3. Statistical Analysis for Finite Horizon Simulation Models

LEARNING OBJECTIVES

- To be able to recognize which type of planning horizon is applicable to a simulation study
- To be able to recognize the different types of statistical quantities used within and produced by the execution of simulation models
- To review statistical concepts and apply them to the analysis of finite horizon simulation studies
- To be able to analyze finite horizon simulations via the method of replications
- Introduce new Arena modeling constructs and how to use them in more complex modeling situations
- To be able to analyze and compare the results from two or more design configuration via simulation

As discussed in Chapter 1 and seen in Chapter 2 discrete event simulation is a very useful tool when randomness is an important aspect of the system environment and thus needs to be included in the simulation modeling. Because the inputs to the simulation are random, the outputs from the simulation are also random. This concept is illustrated in Figure 3.1. 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, specifically that of finite horizon simulations.

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.

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 will also discuss 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

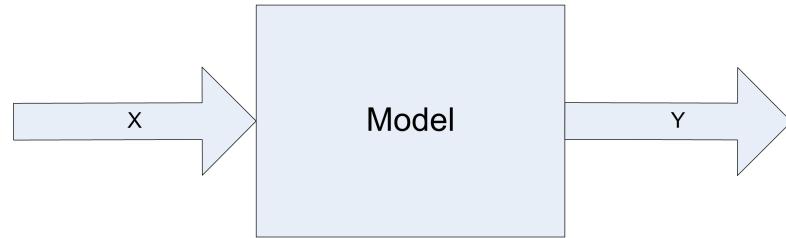


Figure 3.1.: Random input implies random output

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. Before addressing these important concepts, we need to understand how the time horizon of a simulation study is determined and how it might affect the statistical methods that will be required during the analysis.

3.1. Finite versus Infinite Horizon Simulation Studies

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 define 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 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

3.1. Finite versus Infinite Horizon Simulation Studies

process. 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 9am and close at 5pm
- 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.

For a finite horizon situation, we want to observe the behavior of the system over the specified time horizon and report on its performance. The observations available over one time horizon create a sample of performance. This sample of performance is sometimes referred to as a sample path because it is based on the trajectory of the performance measures over time. When we repeatedly run the simulation over multiple instances of the time horizon, we are observing different samples (sample paths). If each of the generated sample paths are independent and identically distributed, we call an instance a *replication*, and the set of instances as *replications*. The length of each sample path or *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 beginning point or ending point. In the case of infinite horizon simulation studies, we are almost always interested in reporting on the long-run performance of the system. Thus, the desired starting conditions of interest are sometime in the far off future. In addition, because we want long run performance there is no natural ending point of interest. Thus, we consider the horizon infinite. 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

3. Statistical Analysis for Finite Horizon Simulation Models

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 other words, 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 more typical type of simulation 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 finite horizon simulation studies will be discussed in this chapter while the discussion of infinite horizon simulation is presented in Chapter 5.

In the next section, we will examine in more detail the concepts of samples, sample paths, and replications by defining the types of statistical quantities observed within a simulation.

3.2. Types of Statistical Quantities in Simulation

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. This execution generates a sample path of the behavior of the system. During the execution of the simulation, the generated sample path 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. As previously noted, 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 1.2 illustrates the concept of replications being repeated (independent) sample paths.

3.2. Types of Statistical Quantities in Simulation

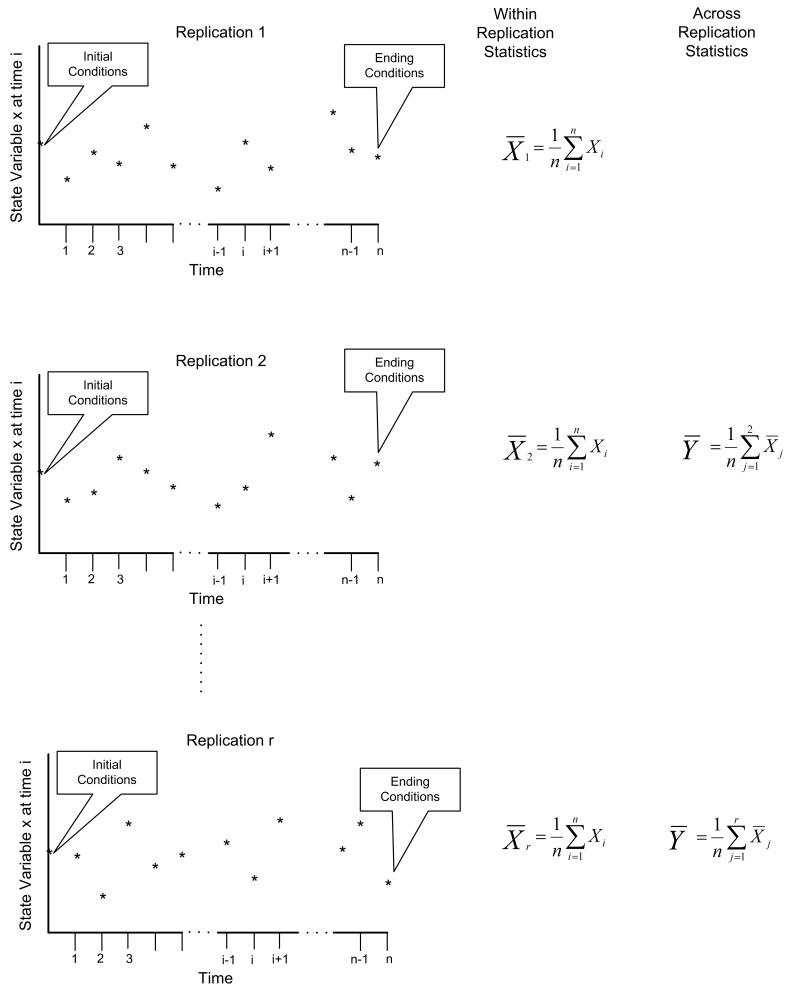


Figure 3.2.: The concept of replicated sample paths

3.2.1. Within Replication Observations

Within a single sample path (replication), the statistical behavior of the model can be observed. The statistical quantities collected during a replication are called *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 single sample path execution. The observations used to form within replication statistics are not likely to be independent and identically distributed.

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

3. Statistical Analysis for Finite Horizon Simulation Models

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 when it occurred, 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. The vast majority of tally based data concerns the entities that flow through the system.

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.

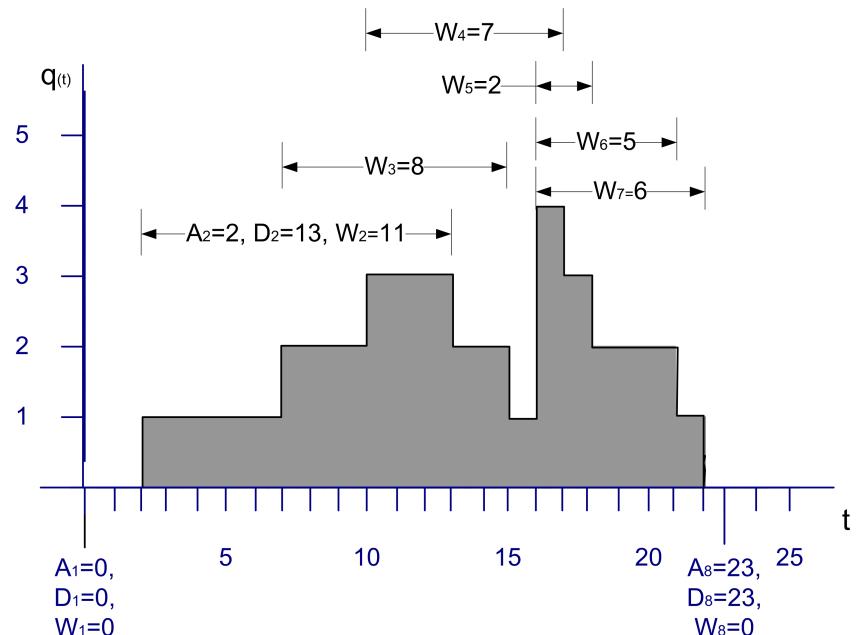


Figure 3.3.: Sample path for tally and time-persistent data

Figure 3.3 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. Let's define some variables based on the data within Figure 3.3.

- 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

3.2. Types of Statistical Quantities in Simulation

- 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, etc. 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 in Chapter 5.

The other type of statistical quantity 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 3.3, $q(t)$ 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

$$\int_a^b f(x)dx = f(c)(b - a)$$

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 or time-weighted average:

$$\bar{L}_q(n) = \frac{1}{t_n - t_0} \int_{t_0}^{t_n} q(t)dt$$

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).

3. Statistical Analysis for Finite Horizon Simulation Models

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. Let's take a look at an example.

Example 3.1. Consider Figure 3.3, which shows the changes in queue length over a simulated period of 25 time units. Compute the time average number in queue over the interval of time from 0 to 25. Compute the percentage of time that the queue had $\{0, 1, 2, 3, 4\}$ customers.

3.2. Types of Statistical Quantities in Simulation

Reviewing Figure 3.3 indicates the changes 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:

$$\begin{aligned}\hat{p}_0 &= \frac{T_0}{T} = \frac{(2-0) + (25-22)}{25} = \frac{2+3}{25} = \frac{5}{25} = 0.2 \\ \hat{p}_1 &= \frac{T_1}{T} = \frac{(7-2) + (16-15) + (22-21)}{25} = \frac{5+1+1}{25} = \frac{7}{25} = 0.28 \\ \hat{p}_2 &= \frac{T_2}{T} = \frac{3+2+3}{25} = \frac{8}{25} = 0.28 \\ \hat{p}_3 &= \frac{T_3}{T} = \frac{3+1}{25} = \frac{4}{25} = 0.16 \\ \hat{p}_4 &= \frac{T_4}{T} = \frac{1}{25} = \frac{1}{25} = 0.04\end{aligned}$$

Notice that the sum of the \hat{p}_i adds to one. 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 that entered the queue (and had to wait) are shown in the figure.

The examples presented in this section were all based on within replication observations. When we collect statistics on within replication observations, we get within replication statistics. the

3. Statistical Analysis for Finite Horizon Simulation Models

two most natural statistics to collect during a replication are the sample average and the time-weighted average. Arena computes these quantities using the `TAVG(tally ID)` and the `DAVG(dstat ID)` functions. Let's take a closer look at how these functions operate.

We will start with a discussion of collecting statistics for observation-based data.

Let $(x_1, x_2, x_3, \dots, x_{n(t)})$ be a sequence of observations up to and including time t .

Let $n(t)$ be the number of observations up to and including time t .

Let $\text{sum}(t)$ be the cumulative sum of the observations up to and including time t . That is,

$$\text{sum}(t) = \sum_{i=1}^{n(t)} x_i$$

Let $\bar{x}(t)$ be the cumulative average of the observations up to and including time t . That is,

$$\bar{x}(t) = \frac{1}{n(t)} \sum_{i=1}^{n(t)} x_i = \frac{\text{sum}(t)}{n(t)}$$

Similar to the observation-based case, we can define the following notation. Let $\text{area}(t)$ be the cumulative area under the state variable curve, $y(t)$.

$$\text{area}(t) = \int_0^t y(t) dt$$

Define $\bar{y}(t)$ as the cumulative average up to and including time t of the state variable $y(t)$, such that:

$$\bar{y}(t) = \frac{\int_0^t y(t) dt}{t} = \frac{\text{area}(t)}{t}$$

The `TAVG(tally ID)` function will collect $\bar{x}(t)$ and the `DAVG(dstat ID)` function collects $\bar{y}(t)$. The tally ID is the name of the corresponding tally variable, typically defined within the RECORD module and `dstat ID` is the name of the corresponding time-persistent variable defined via a TIME PERSISTENT module. Using the previously defined notation, we can note the following Arena functions:

- `TAVG(tally ID)` = $\bar{x}(t)$
- `TNUM(tally ID)` = $n(t)$.

Thus, we can compute $\text{sum}(t)$ for some tally variable via:

$$\text{sum}(t) = \text{TAVG(tally ID)} * \text{TNUM(tally ID)}$$

We have the ability to observe $n(t)$ and $\text{sum}(t)$ via these functions during a replication and at the end of a replication.

To compute the time-persistent averages within Arena we can proceed in a very similar fashion as previously described by using the `DAVG(dstat ID)` function. The `DAVG(dstat ID)` function is $\bar{y}(t)$ the cumulative average of a time-persistent variable. Thus, we can easily compute $\text{area}(t) = t \times \bar{y}(t)$ via `DAVG(dstat ID)^*TNOW`. The special variable `TNOW` provides the current time of the simulation.

Thus, at the end of a replication we can observe `TAVG(tally ID)` and `DAVG(dstat ID)` giving us one observation per replication for these statistics. This is exactly what Arena does automatically when it reports the Category Overview Statistics, which are, in fact, averages across replications.

3.2.2. Across Replication Observations

The statistical quantities collected across replications are called *across replication statistics*. *Across replication statistics* are collected based on the observation of the final value of a within replication statistic. Since across replication statistics are formed from the final value of a within replication statistic (e.g. `TAVG(tally ID)` or `DAVG(dstat ID)`), one observation per replication for each performance measure is available. Since each replication is considered independent, the observations that form the sample of an across replication statistic should be independent and identically distributed. As previously noted, the observations that form the sample path for within replication statistics are, in general, neither independent nor identically distributed. Thus, 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.

Finite horizon simulations can be analyzed by traditional statistical methodologies that assume a random sample, i.e. independent and identically distributed random variables by using across replication observations. 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, i.e. 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:

3. Statistical Analysis for Finite Horizon Simulation Models

$$\bar{Y}_r = \frac{1}{m_r} \sum_{j=1}^{m_r} Y_{rj}$$

If the data are time-based then,

$$\bar{Y}_r = \frac{1}{T_E} \int_0^{T_E} Y_r(t) dt$$

\bar{Y}_r is the sample average based on the observation within the r^{th} replication. Arena observes \bar{Y}_r via the `TAVG()` and `DAVG()` functions at the end of every replication for every performance measure. Thus, \bar{Y}_r is a random variable that is observed at the end of each replication, therefore, \bar{Y}_r for $r = 1, 2, \dots, n$ forms a random sample, $(\bar{Y}_1, \bar{Y}_2, \dots, \bar{Y}_n)$. This sample holds *across replication observations* and statistics computed on these observations are called *across replication statistics*. Because across replication observations are independent and identically distributed, they form a random sample and therefore standard statistical analysis techniques can be applied. Another nice aspect of the across replication observations, $(\bar{Y}_1, \bar{Y}_2, \dots, \bar{Y}_n)$, is the fact that they are often the average of many within replication observations. Because of the central limit theorem, the sampling distribution of the average tends to be normally distributed. This is good because many statistical analysis techniques assume that the random sample is normally distributed.

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, $W_{rj}, j = 1, \dots, m_r$, represents 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, i.e. 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 that are averaged to form across replication observations.

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, i.e. you are interested in $\theta = P\{N(t = 5pm) > 0\}$. 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 at the end of the replication. For this situation, an indicator variable, I_r , for each replication can be defined as follows:

$$I_r = \begin{cases} 1 & N(t = 5pm) > 0 \\ 0 & N(t = 5pm) \leq 0 \end{cases}$$

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. We can use the observations of the indicator variable to estimate the desired probability. Technically, the observation of I_r occurs within (at the end) of the replication. The collection of the I_r for $r = 1, 2, \dots, n$ are the across replication observations and the estimator for the probability:

$$\hat{p} = \frac{1}{n} \sum_{r=1}^n I_r$$

is our across replication statistic. Clearly, in this situation, the sample (I_1, I_2, \dots, I_n) would not be normally distributed; however, the observations would still be independent and identically distributed.

Using the method of replication, the analysis of the system will be based on a random sample. A key decision 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 ± 2 minutes. A review of these and other statistical concepts will be the focus of the next section.

3.3. Review of Statistical Concepts

The simulation models that have been illustrated have estimated the quantity of interest with a point estimate (i.e. \bar{X}). For example, in Example 2.1, we knew that the true area under the curve is 4.66; however, the point estimate returned by the simulation was a value of 4.3872, 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 in order to understand the meaning of sampling error.

3. Statistical Analysis for Finite Horizon Simulation Models

3.3.1. 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 sample, if 1) the X_i are independent random variables and 2) every X_i has the same probability distribution. Let's assume that these two assumptions have been established. Denote the unknown cumulative distribution function 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:

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i \quad (3.1)$$

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:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

A confidence interval expresses a degree of certainty associated with a point estimate. A specific confidence interval does not imply that the parameter μ is inside the interval. In fact, the true parameter is either in the interval or it is 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 μ should be close to $1 - \alpha$. The value α 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 α of making a bad decision. Think of α as the risk that using the confidence interval procedure will get you fired.

3.3. Review of Statistical Concepts

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 then you can form an approximate confidence interval on the point estimate, \bar{x} . Assuming that the sample variance:

$$S^2(n) = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2 \quad (3.2)$$

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

$$\bar{x} \pm t_{1-(\alpha/2), n-1} \frac{s}{\sqrt{n}} \quad (3.3)$$

where $t_{p,\nu}$ is the $100p$ percentage point of the Student t-distribution with ν degrees of freedom. The spreadsheet function T.INV(p, degrees freedom) computes the *left-tailed* Student-t distribution value for a given probability and degrees of freedom. That is, $t_{p,\nu} = T.INV(p, \nu)$. The spreadsheet tab labeled *Student-t* in the spreadsheet that accompanies this chapter illustrates functions related to the Student-t distribution. Thus, in order to compute $t_{1-(\alpha/2), n-1}$, use the following formula:

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

The T.INV.2T(α, ν) spreadsheet function directly computes the *upper two-tailed* value. That is,

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

Within the statistical package R, this computation is straight-forward by using the $t_{p,n} = qt(p, n)$ function.

$$t_{p,n} = qt(p, n)$$

```
#'qt(0.975, 5)
qt(0.975, 5)
```

```
## [1] 2.570582
```

3. Statistical Analysis for Finite Horizon Simulation Models

```
#' set alpha
alpha = 0.05
#'qt(1-(alpha/2), 5)
qt(1-(alpha/2), 5)
```

```
## [1] 2.570582
```

3.3.2. Sample Size Determination

The confidence interval for a point estimator can serve as the basis for determining how many observations to have in the sample. From Equation ((3.3)), the quantity:

$$h = t_{1-(\alpha/2),n-1} \frac{s}{\sqrt{n}} \quad (3.4)$$

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

$$h = t_{1-(\alpha/2),n-1} \frac{s}{\sqrt{n}} \leq \epsilon \quad (3.5)$$

We call ϵ the margin of error for the bound. Unfortunately, $t_{1-(\alpha/2),n-1}$ depends on n , and thus Equation ((3.5)) is an iterative equation. That is, you must try different values of n until the condition is satisfied. Within this text, we call this method for determining the sample size the *Student-T iterative method*.

Alternatively, the required sample size can be approximated using the normal distribution. Solving Equation ((3.5)) for n yields:

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

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

$$n \geq \left(\frac{z_{1-(\alpha/2)} s}{\epsilon} \right)^2 \quad (3.6)$$

Within this text, we refer to this method for determining the sample size as the *normal approximation* method. This approximation 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 either the *normal approximation* method or the *Student-T iterative* method, you must have an initial value for, s , the sample standard deviation. The simplest way to get an

initial estimate of s is to make a small initial pilot sample (e.g. $n_0 = 5$). 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 ϵ can result in very large sample sizes.

Example 3.2. Suppose we are required to estimate the output from a simulation so that we are 99% confidence that we are within ± 0.1 of the true population mean. After taking a pilot sample of size $n = 10$, we have estimated $s = 6$. What is the required sample size?

We will use Equation ((3.6)) to determine the sample size requirement. For a 99% confidence interval, we have $\alpha = 0.01$ and $\alpha/2 = 0.005$. Thus, $z_{1-0.005} = z_{0.995} = 2.576$. Because the margin of error, ϵ is 0.1, we have that,

$$n \geq \left(\frac{z_{1-(\alpha/2)} s}{\epsilon} \right)^2 = \left(\frac{2.576 \times 6}{0.1} \right)^2 = 23888.8 \approx 23889$$

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

$$\hat{p} \pm z_{1-(\alpha/2)} \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \quad (3.7)$$

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

$$n = \left(\frac{z_{1-(\alpha/2)}}{\epsilon} \right)^2 \hat{p}(1-\hat{p}) \quad (3.8)$$

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

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. Now, let's illustrate these methods based on a small Arena simulation.

3.3.3. Determining the Sample Size for an Arena Simulation

To facilitate some of the calculations related to determining the sample size for a simulation experiment, I have constructed a spreadsheet called *SampleSizeDetermination.xlsx*, which is found in the book support files for this chapter. You may want to utilize that spreadsheet as you go through this and subsequent sections.

Using a simple example, we will illustrate how to determine the sample size necessary to estimate a quantity of interest with a high level of confidence.

Example 3.3. Suppose that we want to simulate a normally distributed random variable, X , with $E[X] = 10$ and $Var[X] = 16$. From the simulation, we want to estimate the true population mean with 99 percent confidence for a half-width ± 0.50 margin of error. In addition to estimating the population mean, we want to estimate the probability that the random variable exceeds 8. That is, we want to estimate, $p = P[X > 8]$. We want to be 95% confident that our estimate of $P[X > 8]$ has a margin of error of ± 0.05 . What are the sample size requirements needed to meet the desired margins of error?

Using simulation for this example is for illustrative purposes. Naturally, we actually know the true population mean and we can easily compute the $p = P[X > 8]$. for this situation. However, the example will illustrate sample size determination, which is an important planning requirement for simulation experiments, and the example reinforces how to use the RECORD module.

Let X represent the unknown random variable of interest. Then, we are interested in estimating $E[X] = \mu$ and $p = P[X > 8]$. 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 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. To estimate p , define the following indicator variable:

$$Y_i = \begin{cases} 1 & X_i > 8 \\ 0 & X_i \leq 8 \end{cases}$$

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 $X_i > 8$, 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 $X_i > 8$ occurred in the sample. Call this $\#\{X_i > 8\}$. 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:

$$\hat{p} = \bar{Y} = \frac{1}{n} \sum_{i=1}^n Y_i = \frac{\#\{X_i > 8\}}{n}$$

Therefore, computing the average of an indicator variable will estimate the desired probability. The pseudo-code for this situation is quite simple:

```

CREATE 1 entity
ASSIGN myX = NORM(10, 4)
Begin RECORD
    myX as EstimatedX
    (myX > 8) as EstimatedProb
End RECORD
DISPOSE

```

In the pseudo-code, the RECORD module is recording observations of the quantity ($myX > 8$). This quantity is actually a logical condition which will evaluate to true (1) or false (0) given the value of myX . Thus, the RECORD module will be recording observations of 1's and 0's. Because the RECORD module computes that average of the values that it "records", we will get an estimate of the probability. Thus, the quantity, ($myX > 8$) is an indicator variable for the desired event. Using a RECORD module to estimate a probability in this manner is quite effective.

Now, in more realistic simulation situations, we would not know the true population mean and variance. Thus, in solving this problem, we will ignore that information. To apply the previously discussed sample size determination methods, we need estimates of $\sigma = \sqrt{Var[X]}$ and $p = P[X > 8]$. In order to get estimates of these quantities, we need to run our simulation experiment; however, in order to run our simulation experiment, we need to know how many observations to make. This is quite a catch-22 situation!

To proceed, we need to run our simulation model for a pilot experiment. From a small number of samples (called a pilot experiment, pilot run or pilot study), we will estimate σ and $p = P[X > 8]$ and then use those estimates to determine how many samples we need to meet the desired margins of error. We will then re-run the simulation using the recommended sample size.

The natural question to ask is how big should your pilot sample be? In general, this depends on how much it costs for you to collect the sample. Within a simulation experiment context, generally, cost translates into how long your simulation model takes to execute. For this small problem, the execution time is trivial, but for large and complex simulation models the execution time may be in hours or even days. A general rule of thumb is between 10 and 30 observations. For this example, we will use an initial pilot sample size, $n_0 = 20$.

The Arena model can be easily built by following the pseudo-code. You can find the Arena model as part of the files associated with the chapter. Running the model for the 20 replications yields the following results.

3. Statistical Analysis for Finite Horizon Simulation Models

Table 3.1.: Arena results for $n_0 = 20$ replications

Performance Measures	Average	95% Half-Width
EstimatedProb	0.70	0.22
EstimatedX	10.3224	2.24

Notice that the confidence interval for the true mean is $(8.0824, 12.5624)$ with a half-width of 2.24, which exceeds the desired margin of error, $\epsilon = 0.5$. Notice also that this particular confidence interval happens to contain the true mean $E[X] = 10$. To determine the sample size so that we get a half-width that is less than $\epsilon = 0.1$ with 99% confidence, we can use Equation (3.6). In order to do this, we must first have an estimate of the sample standard deviation, s . Since Arena reports a half-width for a 95% confidence interval, we need to use Equation (3.4) and solve for s in terms of h . Rearranging Equation (3.4) yields,

$$s = \frac{h\sqrt{n}}{t_{1-(\alpha/2), n-1}} \quad (3.9)$$

Using the results from Table 3.1 and $\alpha = 0.05$ in Equation (3.9) yields,

$$s_0 = \frac{h\sqrt{n_0}}{t_{1-(\alpha/2), n_0-1}} = \frac{2.24\sqrt{20}}{t_{0.975, 19}} = \frac{2.24\sqrt{20}}{2.093} = 4.7862$$

Now, we can use Equation (3.6) to determine the sample size requirement. For a 99% confidence interval, we have $\alpha = 0.01$ and $\alpha/2 = 0.005$. Thus, $z_{0.995} = 2.576$. Because the margin of error, ϵ is 0.5, we have that,

$$n \geq \left(\frac{z_{1-(\alpha/2)} s}{\epsilon} \right)^2 = \left(\frac{2.576 \times 4.7862}{0.5} \right)^2 = 608.042 \approx 609$$

To determine the sample size for estimating $p = P[X > 8]$ with 95% confidence to ± 0.1 , we can use Equation (3.8)

$$n = \left(\frac{z_{1-(\alpha/2)}}{\epsilon} \right)^2 \hat{p}(1-\hat{p}) = \left(\frac{z_{0.975}}{0.1} \right)^2 (0.22)(1-0.22) = \left(\frac{1.96}{0.1} \right)^2 (0.22)(0.78) = 65.92 \approx 66$$

By using the maximum of $\max(66, 609) = 609$, we can re-run the simulation this number of replications. Doing so, yields,

Table 3.2.: Arena Results for $n = 609$ Replications

Performance Measures	Average	95% Half-Width
EstimatedProb	0.69	0.04
EstimatedX	9.9032	0.32

3.3. Review of Statistical Concepts

As can be seen in Table 3.2, the half-width values meet the desire margins of error. It may be possible that the margins of error might not be met. This suggests that more than $n = 609$ observations is needed to meet the margin of error criteria. Equation (3.6) and Equation (3.8) are only approximations and based on a pilot sample. Thus, if there was considerable sampling error associated with the pilot sample, the approximations may be inadequate.

As can be noted from this example in order to apply the normal approximation method for determining the sample size based on the pilot run of an Arena model, we need to compute the initial sample standard deviation, s_0 , from the initial reported half-width, h , that appears on the Arena report. This requires the use of Equation (3.9) to first compute the value of s from h . We can avoid this calculation by using the *half-width ratio* method for determining the sample size.

Let h_0 be the initial value for the half-width from the pilot run of n_0 replications. Then, rewriting Equation (3.4) in terms of the pilot data yields:

$$h_0 = t_{1-(\alpha/2), n_0-1} \frac{s_0}{\sqrt{n_0}}$$

Solving for n_0 yields:

$$n_0 = t_{1-(\alpha/2), n_0-1}^2 \frac{s_0^2}{h_0^2} \quad (3.10)$$

Similarly for any n , we have:

$$n = t_{1-(\alpha/2), n-1}^2 \frac{s^2}{h^2} \quad (3.11)$$

Taking the ratio of n_0 to n (Equations (3.10) and (3.11)) and assuming that $t_{1-(\alpha/2), n-1}$ is approximately equal to $t_{1-(\alpha/2), n_0-1}$ and s^2 is approximately equal to s_0^2 , yields,

$$n \cong n_0 \frac{h_0^2}{h^2} = n_0 \left(\frac{h_0}{h} \right)^2$$

This is the half-width ratio equation.

$$n \geq n_0 \left(\frac{h_0}{h} \right)^2 \quad (3.12)$$

The issue that we face when applying Equation (3.12) is that Arena only reports the 95% confidence interval half-width on its reports. Thus, to apply Equation (3.12) the value of h_0 will be based on an $\alpha = 0.05$. If the desired half-width, h , is required for a different confidence level, e.g. a 99% confidence interval, then you must first translate h_0 to be based on the desired confidence level. This requires computing s from Equation (3.9) and then recomputing the actual half-width, h , for the desired confidence level.

3. Statistical Analysis for Finite Horizon Simulation Models

Using the results from Table 3.1 and $\alpha = 0.05$ in Equation (3.9) we already know that $s_0 = 4.7862$ for a 95% confidence interval. Thus, for a 99% confidence interval, where $\alpha = 0.01$ and $\alpha/2 = 0.005$, we have:

$$h_0 = t_{1-(\alpha/2), n_0-1} \frac{s_0}{\sqrt{n_0}} = t_{0.995, 19} \frac{4.7862}{\sqrt{20}} = 2.861 \frac{4.7862}{\sqrt{20}} = 3.0618$$

Now, we can apply Equation (3.12) to this problem with the desire half-width bound $\epsilon = 0.5 = h$:

$$n \geq n_0 \left(\frac{h_0}{h} \right)^2 = 20 \left(\frac{3.0618}{h} \right)^2 = 20 \left(\frac{3.0618}{0.5} \right)^2 = 749.98 \cong 750$$

We see that for this pilot sample, the half-width ratio method recommends a substantially large sample size, 750 versus 609. In practice, the half-width ratio method tends to be conservative by recommending a larger sample size. We will see another example of these methods within the next section.

Based on these examples, you should have a basic understanding of how simulation can be performed to meet a desired margin of error. In general, simulation models are much more interesting than this simple example. To deepen your knowledge, we will model a more interesting situation in the next section.

3.4. Modeling a STEM Career Mixer

In this section, we will model the operation of a STEM Career Fair mixer during a six-hour time period. The purpose of the example is to illustrate the following concepts:

- Probabilistic flow of entities using the DECIDE module with the by chance option
 - Additional options of the PROCESS module
 - Using the Label module to direct the flow of entities
 - Collecting statistics on observational (tally) data using the RECORD module
 - Collecting statistics on time-persistent data using the Statistics panel
 - Analysis of a finite horizon model to ensure a pre-specified half-width requirement
-

Example 3.4. Students arrive to a STEM career mixer event according to a Poisson process at a rate of 0.5 student per minute. The students first go to the name tag station, where it takes between 10 and 45 seconds uniformly distributed to write their name and affix the tag to themselves. We assume that there is plenty of space at the tag station, as well has plenty of tags and markers, such that a queue never forms.

After getting a name tag, 50% of the students wander aimlessly around, chatting and laughing with their friends until they get tired of wandering. The time for aimless students to wander around is triangularly distributed with a minimum of 10 minutes, a most likely value of 15 minutes, and a maximum value of 45 minutes. After wandering aimlessly, 90% decide to buckle down and visit companies and the remaining 10% just are too tired or timid and just leave. Those that decide to buckle down visit the MalWart station and then the JHBunt station as described next.

The remaining 50% of the original, newly arriving students (call them non-aimless students), first visit the MalWart company station where there are 2 recruiters taking resumes and chatting with applicants. At the MalWart station, the student waits in a single line (first come first served) for 1 of the 2 recruiters. After getting 1 of the 2 recruiters, the student and recruiter chat about the opportunities at MalWart. The time that the student and recruiter interact is exponentially distributed with a mean of 3 minutes. After visiting the MalWart station, the student moves to the JHBunt company station, which is staffed by 3 recruiters. Again, the students form a single line and pick the next available recruiter. The time that the student and recruiter interact at the JHBunt station is also exponentially distribution, but with a mean of 6 minutes. After visiting the JHBunt station, the student departs the mixer.

The organizer of the mixer is interested in collecting statistics on the following quantities within the model:

- number of students attending the mixer at any time t
- number of students wandering at any time t
- utilization of the recruiters at the MalWart station
- utilization of the recruiters at the JHBunt station
- number of students waiting at the MalWart station
- number of students waiting at the JHBunt station
- the waiting time of students waiting at the MalWart station
- the waiting time of students waiting at the JHBunt station
- total time students spend at the mixer broken down in the following manner
 - all students regardless of what they do and when they leave

3. Statistical Analysis for Finite Horizon Simulation Models

- students that wander and then visit recruiters
- students that do not wander

The STEM mixer organizer is interested in estimating the average time students spend at the mixer (regardless of what they do) to within plus or minus 1 minute with a 95% confidence level.

3.4.1. Conceptualizing the System

When developing a simulation model, whether you are an experienced analyst or a novice, you should follow a modeling recipe. I recommend developing answers to the following questions:

- *What is the system?*
 - *What are the elements of the system?*
 - *What information is known by the system?*
- *What are the required performance measures?*
- *What are the entity types?*
 - *What information must be recorded or remembered for each entity instance?*
 - *How are entities (entity instances) introduced into the system?*
- *What are the resources that are used by the entity types?*
 - *Which entity types 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*

We will apply each of these questions to the STEM mixer example. The first set of questions: *What is the system?* *What information is known by the system?* are used to understand what should be included in the modeling and what should not be included. In addition, understanding the system to be modeled is essential for validating your simulation model. If you have not clearly defined what you are modeling (i. e. the system), you will have a very difficult time validating your simulation model.

The first thing that I try to do when attempting to understand the system is to draw a picture. A hand drawn picture or sketch of the system to be modeled is useful for a variety of reasons. First, a drawing attempts to put down “on paper” what is in your head. This aids in making the modeling more concrete and it aids in communicating with system stakeholders. In fact,

a group stakeholder meeting where the group draws the system on a shared whiteboard helps to identify important system elements to include in the model, fosters a shared understanding, and facilitates model acceptance by the potential end-users and decision makers.

You might be thinking that the system is too complex to sketch or that it is impossible to capture all the elements of the system within a drawing. Well, you are probably correct, but drawing an idealized version of the system helps modelers to understand *that you do not have to model reality* to get good answers. That is, drawing helps to *abstract* out the important and essential elements that need to be modeled. In addition, you might think, why not take some photographs of the system instead of drawing? I say, go ahead, and take photographs. I say, find some blueprints or other helpful artifacts that represent the system and its components. These kinds of things can be very helpful if the system or process already exists. However, do not stop at photographs, drawing facilitates free flowing ideas, and it is an active/engaging process. Do not be afraid to draw. The art of abstraction is essential to good modeling practice.

Alright, have I convinced you to make a drawing? So, your next question is, what should my drawing look like and what are the rules for making a drawing? Really? My response to those kinds of questions is that you have not really bought into the benefits of drawing and are looking for reasons to not do it. There are no concrete rules for drawing a picture of the system. I like to suggest that the drawing can be like one of your famous kindergarten pictures you used to share with your grandparents. In fact, if your grandparents could look at the drawing and be able to describe back to you what you will be simulating, you will be on the right track! There are not really any rules.

Well, I will take that back. I have one rule. The drawing should not look like an engineer drew it. Try not to use engineering shapes, geometric shapes, finely drawn arrows, etc. You are not trying to make a blueprint. You are not trying to reproduce the reality of the system by drawing it to perfect scale. You are attempting to conceptualize the system in a form that facilitates communication. Don't be afraid to put some labels and text on the drawing. Make the drawing a picture that is rich¹ in the elements that are in the system. Also, the drawing does not have to be perfect, and it does not have to be complete. Embrace the fact that you may have to circle back and iterate on the drawing as new modeling issues arise. You have made an excellent drawing if another simulation analyst could take your drawing and write a useful narrative description of what you are modeling.

Figure 3.4 illustrates a drawing for the STEM mixer problem. As you can see in the drawing, we have students arriving to a room that contains some tables for conversation between attendees. In addition, the two company stations are denoted with the recruiters and the waiting students. We also see the wandering students and the timid students' paths through the system. At this point, we have a narrative description of the system (via the problem statement) and a useful drawing of the system. We are ready to answer the following questions:

- *What are the elements of the system?*
- *What information is known by the system?*

¹<http://www.msppguide.org/tool/rich-picture>

3. Statistical Analysis for Finite Horizon Simulation Models

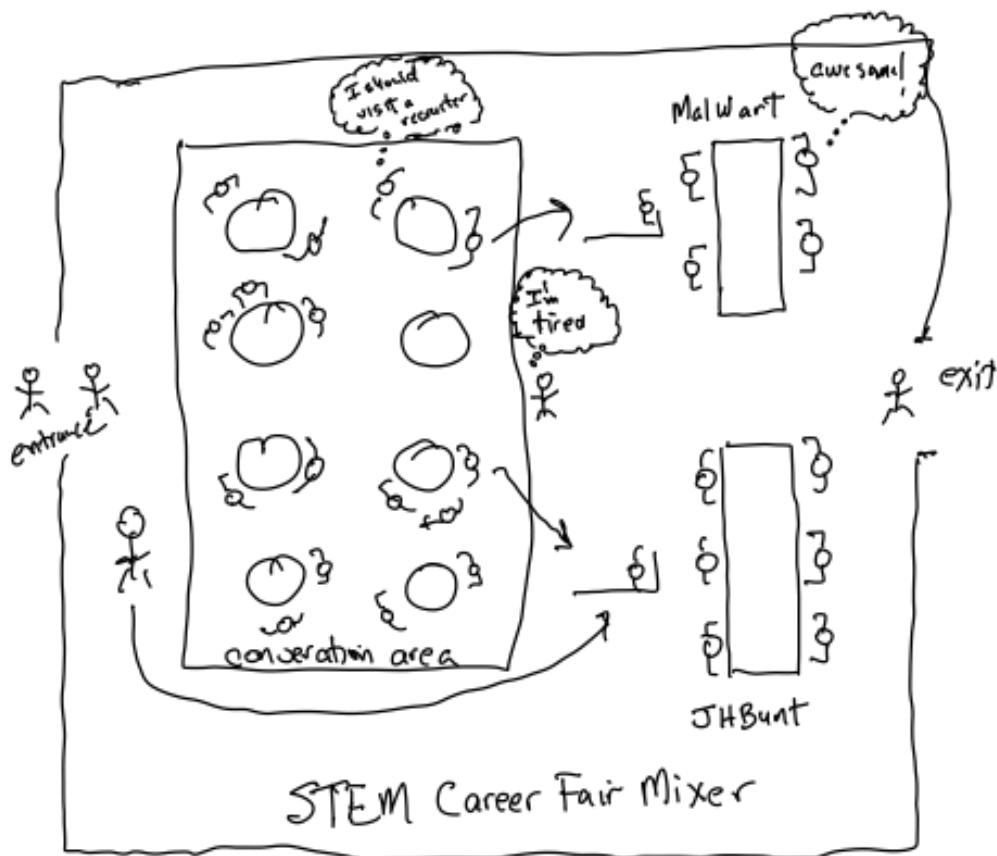


Figure 3.4.: Rich picture system drawing of STEM Career Mixer

When answering the question “what are the elements?”, your focus should be on two things: 1) the concrete structural things/objects that are required for the system to operate and 2) the things that are operated on by the system (i.e. the things that flow through the system).

What are the elements?

- students (non-wanderers, wanderers, timid)
- recruiters: 2 for MalWart and 3 for JHBunt
- waiting lines: one line for MalWart recruiters, one line for JHBunt recruiters
- company stations: MalWart and JHBunt
- The paths that students may take.

Notice that the answers to this question are mostly things that we can point at within our picture (or the real system).

When answering the question “what information is known at the system level?”, your focus should be identifying facts, input parameters, and environmental parameters. Facts tend to relate to specific elements identified by the previous question. Input parameters are key variables, distributions, etc. that are typically under the control of the analyst and might likely be varied during the use of the model. Environmental parameters are also a kind of input parameter, but they are less likely to be varied by the modeler because they represent the operating environment of the system that is most likely not under control of the analyst to change. However, do not get hung up on the distinction between input parameters and environmental parameters, because their classification may change based on the objectives of the simulation modeling effort. Sometimes innovative solutions come from questioning whether or not an environmental parameter can really be changed.

What information is known at the system level?

- students arrive according to a Poisson process with mean rate 0.5 students per minute or equivalently the time between arrivals is exponentially distributed with a mean of 2 minutes
- time to affix a name tag ~ UNIF(20, 45) seconds
- 50% of students are go-getters and 50% are wanderers
- time to wander ~ TRIA(10, 15, 45) minutes
- 10% of wanderers become timid/tired, 90% visit recruiters
- number of MalWart recruiters = 2
- time spent chatting with MalWart recruiter ~ EXPO(3) minutes
- number of JHBunt recruiters = 3
- time spent chatting with JHBunt recruiter ~ EXPO(6) minutes

A key characteristic of the answers to this question is that the information is “global”. That is, it is known at the system level. We will need to figure out how to represent the storage of this information when implementing the model within software.

The next question to address is *“What are the required performance measures?”* For this situation, we have been given as part of the problem a list of possible performance measures, mostly related to time spent in the system and other standard performance measures related to queueing systems. In general, you will not be given the performance measures. Instead, you will need to interact with the stakeholders and potential users of the model to identify the metrics that they need in order to make decisions based on the model. Identifying the metrics is a key step in the modeling effort. If you are building a model of an existing system, then you can start with the metrics that stakeholders typically use to characterize the operating performance of the system. Typical metrics include cost, waiting time, utilization, work in process, throughput, and probability of meeting design criteria.

3. Statistical Analysis for Finite Horizon Simulation Models

Once you have a list of performance measures, it is useful to think about how you would collect those statistics *if you were an observer standing within the system*. Let's pretend that we need to collect the total time that a student spends at the career fair and we are standing somewhere at the actual mixer. How would you physically perform this data collection task? In order to record the total time spent by each student, you would need to note when each student arrived and when each student departed. Let A_i be the arrival time of the i^{th} student to arrive and let D_i be the departure time of the i^{th} student. Then, the system time (T) of the i^{th} student is $T_i = D_i - A_i$. How could we keep track of A_i for each student? One simple method would be to write the time that the student arrived on a sticky note and stick the note on the student's back. Hey, this is pretend, right? Then, when the student leaves the mixer, you remove the sticky note from their back and look at your watch to get D_i and thus compute, T_i . Easy! We will essentially do just that when implementing the collection of this statistic within the simulation model. Now, consider how you would keep track of the number of students attending the mixer. Again, standing where you can see the entrance and the exit, you would increment a counter every time a student entered and decrement the counter every time a student left. We will essentially do this within the simulation model.

Thus, identifying the performance measures to collect and how you will collect them will answer the 2nd modeling question. You should think about and document how you would do this for every key performance measure. However, you will soon realize that simulation modeling languages, like Arena, have specific constructs that automatically collect common statistical quantities such as resource utilization, queue waiting time, queue size, etc. Therefore, you should concentrate your thinking on those metrics that will not automatically be collected for you by the simulation software environment. In addition to identifying and describing the performance measures to be collected, you should attempt to classify the underlying data needed to compute the statistics as either observation-based (tally) data or time-persistent (time-weighted) data. This classification will be essential in determining the most appropriate constructs to use to capture the statistics within the simulation model. The following table summarizing the type of each performance measure requested for the STEM mixer simulation model.

Performance Measure	Type
Average number of students attending the mixer at any time t	Time persistent
Average number of students wandering within the mixer at any time t	Time persistent
Average utilization of the recruiters at the MalWart station	Time persistent
Average utilization of the recruiters at the JHBunt station	Time persistent
Average number of students waiting at the MalWart station	Time persistent
Average number of students waiting at the JHBunt station	Time persistent
Average waiting time of students waiting at the MalWart station	Tally
Average waiting time of students waiting at the JHBunt station	Tally
Average system time for students regardless of what they do	Tally
Average system time for students that wander and then visit recruiters	Tally
Average system time for students that do not wander	Tally

Now, we are ready to answer the rest of the questions. When performing a process-oriented simulation, it is important to identify the entity types and the characteristics of their instances. An entity type is a classification or indicator that distinguishes between different entity instances. If you are familiar with object-oriented programming, then an entity type is a class, and an entity instance is an object of that class. Thus, an entity type describes the entity instances (entities) that are in its class. An entity instance (or just entity) is something that flows through the processes within the system. Entity instances (or just entities) are realizations of objects within the entity type (class). Different entity types may experience different processes within the system. The next set of questions are about understanding entity types and their instances: *What are the entity types? What information must be recorded or remembered for each entity (entity instance) of each entity type? How are entities (entity instances) for each entity type introduced into the system?*

Clearly, the students are a type of entity. We could think of having different sub-types of the student entity type. That is, we can conceptualize three types of students (non-wanderers, wanderers, and timid/tired). However, rather than defining three different classes or types of students, we will just characterize one general type of entity called a Student and denote the different types with an attribute that indicates the type of student. An attribute is a named property of an entity type and can have different values for different instances. We will use the attribute, myType, to denote the type of student (1= non-wanders, 2 = wanderers, 3 = timid). Then, we will use this type attribute to help in determining the path that a student takes within the system and when collecting the required statistics.

We are basically told how the students (instances of the student entity type) are introduced to the system. That is, we are told that the arrival process for students is Poisson with a mean rate of 0.5 students per minute. Thus, we have that the time between arrivals is exponential with a mean of 2 minutes. *What information do we need to record or be remembered for each student?* The answer to this question identifies the attributes of the entity. Recall that an attribute is a named property of an entity type which can take on different values for different entity instances. Based on our conceptualization of how to collect the total time in the system for the students, it should be clear that each entity (student) needs to remember the time that the student arrived. That is our sticky note on their backs. In addition, since we need to collect statistics related to how long the different types of students spend at the STEM fair, we need each student (entity) to remember what type of student they are (non-wanderer, wanderer, and timid/tired). So, we can use the, myType, attribute to note the type of the student.

After identifying the entity types, you should think about identifying the resources. Resources are system elements that entity instances need in order to proceed through the system. The lack of a resource causes an entity to wait (or block) until the resource can be obtained. By the way, if you need help identifying entity types, you should identify the things that are waiting in your system. It should be clear from the picture that students wait in either of two lines for recruiters. Thus, we have two resources: MalWart Recruiters and JHBunt Recruiters. We should also note the capacity of the resources. The capacity of a resource is the total number of units of the resource that may be used. Conceptually, the 2 MalWart recruiters are identical. There is no difference between them (as far as noted in the problem statement). Thus, the capacity of the

3. Statistical Analysis for Finite Horizon Simulation Models

MalWart recruiter resource is 2 units. Similarly, the JHBunt recruiter resource has a capacity of 3 units.

In order to answer the “and how” part of the question, I like to summarize the entities, resources and activities experienced by the entity types in a table. Recall that an activity is an interval of time bounded by two events. Entities experience activities as they move through the system. The following table summarizes the entity types, the activities, and the resources. This will facilitate the drawing of an activity diagram for the entity types.

Entity Type	Activity	Resource Used
Student (all)	time to affix a name tag ~ UNIF(20, 45) seconds	None
Student (wanderer)	Wandering time ~ TRIA(10, 15, 45) minutes	None
Student (not timid)	Talk with MalWart ~ EXPO(3) minutes	1 of the 2 MalWart recruiters
Student (not timid)	Talk with JHBunt ~ EXPO(6) minutes	1 of the 3 JHBunt recruiters

Now we are ready to document the processes experienced by the entities. A good way to do this is through an activity flow diagram. As described in the Chapter 2, an activity diagram will have boxes that show the activities, circles to show the queues and resources, and arrows, to show the paths taken. Figure 3.5 presents the activity diagram for this situation.

In Figure 3.5, we see that the students are created according to a time between arrival (TBA) process that is exponentially distributed with a mean of 2 minutes and they immediately experience the activity associated with affixing the name tag. Note that the activity does not require any resources and thus has no queue in front of it. Then, the students follow one of two paths, with 50% becoming “non-wanderers” and 50% becoming “wanderers”. The wandering students, wander aimlessly, for period of time, which is represented by another activity that does not require any resources. Then, the 90% of the students follow the non-wanderer path and the remaining 10% of the students, the timid/tired students, leave the system. The students that decide to visit the recruiting stations, first visit the MalWart station, where in the diagram we see that there is an activity box for their talking time with the recruiter. During this talking time, they require one of the recruiters and thus we have a resource circle indicating the usage of the resource. In addition, we have a circle with a queue denoted before the activity to indicate that the students visiting the station may need to wait. The JHBunt station is represented in the same fashion. After visiting the JHBunt station, the students depart the mixer. Now, we are ready to represent this situation using some pseudo-code.

Pseudo-code is simply a computer language like construct for representing the programming elements of a computer program. In this text, we will use pseudo-code that has elements of Arena-like syntax to help document the coding logic, before we use Arena. Let me state that again. Write pseudo-code, before opening up Arena and laying down Arena modules. Why? Pseudo-code facilitates communication of the logic, without a person having to know Arena. Often, other simulation programming environments will have constructs similar to those available within Arena. Thus, your coding logic is more generic. Secondly, pseudo-code is compact

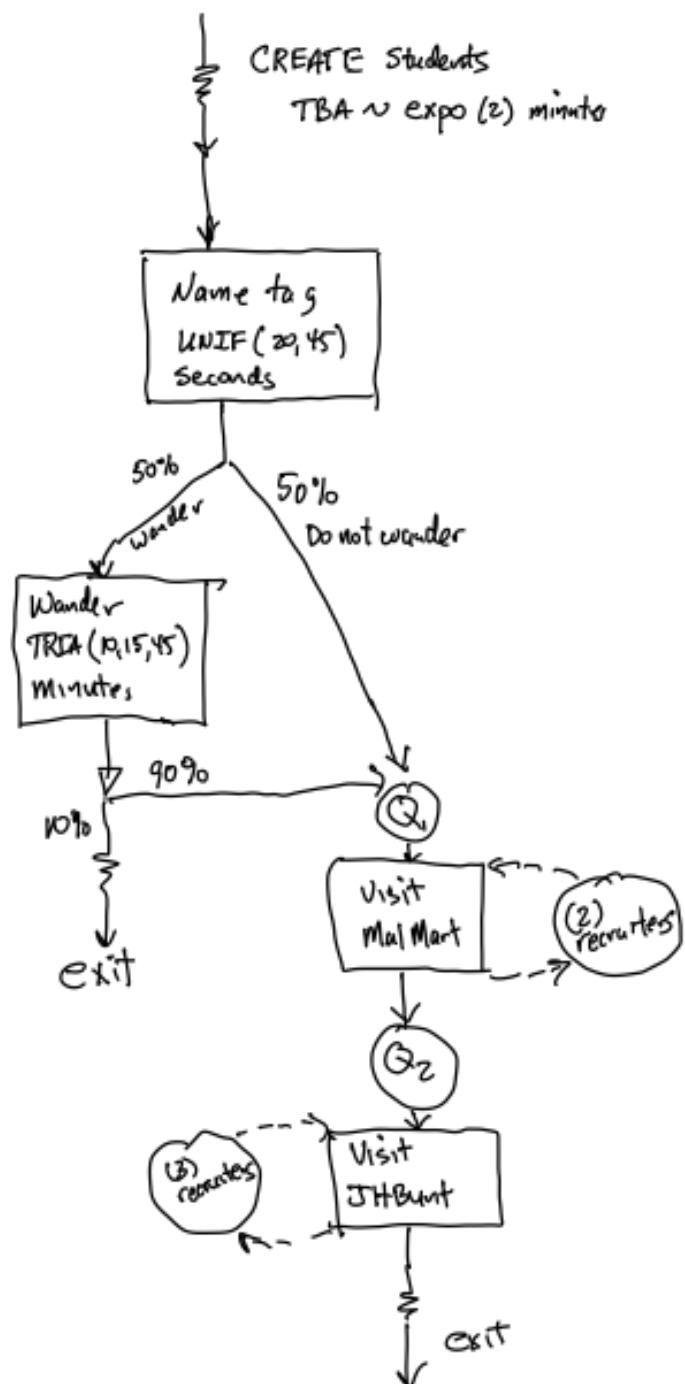


Figure 3.5.: Activity diagram for STEM Career Mixer

3. Statistical Analysis for Finite Horizon Simulation Models

and concise. Lastly, having pseudo-code before using Arena has proven over many years to result in better organized models that tend to work correctly.

The following table is an initial listing of pseudo-code syntax that we will utilize within this example. Notice that in the table, I have noted some of the naming conventions that I recommend and some comments about the usage of the construct.

Keyword	Purpose	Comments
ATTRIBUTES	To provide a list of attributes and their usage	Attribute names should start with "my"
VARIABLES	To provide a list of variables and their usage	Variable names should start with "v"
RESOURCES	To provide a list of resources and their capacity	Resource names should start with "r"
CREATE	To define a creational pattern	Indicate what is being created and the timing of the creation
ASSIGN	To represent the assignment of a value to a variable, attribute, or other construct	BEGIN ASSIGN Multiple assignment statements END ASSIGN
DECIDE IF	To represent conditional logic or path choice	DECIDE IF (condition) Executed if condition is true ELSE Executed if condition is false END DECIDE
DECIDE with chance	To represent probabilistic path choice	DECIDE with chance: p: Executed with probability p 1-p: Executed with probability 1-p END DECIDE
DELAY	To represent a scheduled passage of time for an activity	Indicate the time or distribution associated with the delay and name of activity
SEIZE	To represent the need for a resource	Indicate the name of the resource and the number of units required
RELEASE	To represent the release of units of a resource	Indicate the name of the resource and the number of units released

Keyword	Purpose	Comments
GOTO: Label	To direct the execution to a particular label	Provide the name of the label
Label: Name	To indicate a label where to go	Provide the name of the label
RECORD (expression)	To indicate the collection of statistics	BEGIN RECORD Multiple expression statements END RECORD
DISPOSE	To represent the disposal of the entity that was executing the code	Implies the release of memory associated with the entity

These pseudo-code constructs will represent specific modules or programming constructs within the Arena environment. There are a lot more possibilities and variations of these constructs that will be explored throughout the textbook. One such variation is in describing a process. The SEIZE, DELAY, RELEASE combination is so prevalent, you will see that Arena has defined a PROCESS module that provides common variations of usage of these three constructs. Syntactically, we might indicate the usage of a PROCESS module as follows:

```
BEGIN PROCESS: SEIZE, DELAY, RELEASE
  Resource: worker
  Amount needed: 1
  Delay time: 5 minutes
END PROCESS
```

This is conceptually the same as writing:

```
SEIZE 1 worker
DELAY for 5 minutes
RELEASE 1 worker
```

You decide which you prefer to write. Either case indicates to me that we would use the Arena PROCESS module. I tend to write my pseudo-code in the lowest level of syntax and without worrying about matching perfectly to Arena constructs. The model translation and building phase of the recipe will take care of those worries.

If you have made a rigorous effort to answer the questions and follow the suggested modeling recipe, then the pseudo-code will be a more straightforward exercise than if you did not follow the recipe.

3. Statistical Analysis for Finite Horizon Simulation Models

Students often ask, “how do you come up with the pseudo-code so easily?”. The answer is that I put in the effort before starting the pseudo-code and I point out that they are seeing the end result. That is, there may be a number of iterations of the pseudo-code before I feel that it is a satisfactory representation of what I will need to build the simulation model. In addition, the pseudo-code does not have to be perfect. You should attempt to capture the essence of what needs to be done and not worry that you have an exact listing for the model building stage.

To start the pseudo-code, it is useful to define some of the structural modeling elements that you will use throughout the code such as the attributes, variables, and resources.

Pseudo-Code Definition Elements

ATTRIBUTES:

```
myType // represents the type of the student (1= non-wanders, 2 =  
wanderers, 3 = timid).  
myArrivalTime // the time that the student arrived to the mixer
```

VARIABLES:

```
vNumInSys // represents the number of students attending the mixer at  
any time t
```

RESOURCES:

```
rMalWartRecruiters, 2 // the MalWart recruiters with capacity 2  
rJHBuntRecruiters, 3 // the JHBunt recruiters with capacity 3
```

Considering the pseudo-code for the STEM Fair Mixer example problem, we see the use of ATTRIBUTES, VARIABLES, and RESOURCES to define some elements that will be used throughout the code. Notice the use of “my” and “v” and “r” in the naming of the attributes, variables, and resources.

After defining some of the structural elements needed within the pseudo-code, I then start with introducing the entities into the simulation. That is, I start with the CREATE modules. In addition, I use the activity diagram to assist in determining the logical flow of the pseudo-code and to consider the constructs that will be needed to represent that logical flow.

Pseudo-Code Modeling Elements

```
CREATE students every EXPO(2) minutes with the first arriving at time  
EXPO(2)  
BEGIN ASSIGN  
    myArrivalTime = TNOW // save the current time in the myArrivalTime attribute  
    vNumInSys = vNumInSys + 1  
END ASSIGN  
DELAY UNIF(15,45) seconds to get name tag
```

```

DECIDE with chance:
50%:
  ASSIGN: myType = 1 // non-wandering student
  Goto: Recruiting Stations
50%:
  ASSIGN: myType = 2 // wandering student
  Goto: Wandering
END DECIDE

Label: Wandering
DELAY TRIA(15, 20, 45) minutes for wandering
DECIDE with chance:
10%: Goto: Recruiting Stations
90%:
  ASSIGN: myType = 3 // timid student
  Goto: Exit Mixer
END DECIDE

Label: Recruiting Stations
SEIZE 1 MalWart recruiter
DELAY for EXPO(3) minutes for interaction
RELEASE 1 MalWart recruiter
SEIZE 1 JHBunt recruiter
DELAY for EXPO(6) minutes for interaction
RELEASE 1 JHBunt recruiter
Goto: Exit Mixer

Label: Exit Mixer
ASSIGN: vNumInSys = vNumInSys - 1
ASSIGN: mySysTime = TNOW - myArrivalTime
RECORD mySysTime, as System Time regardless of type
DECIDE IF (myType == 1)
  RECORD mySysTime, as Non-Wandering System Time
ELSE IF (myType == 2)
  RECORD mySysTime, as Wandering System Time
ELSE // myType == 3
  RECORD mySysTime, as Timid Student System Time
END DECIDE
DISPOSE

```

The modeling code starts off with a CREATE statement that indicates the Poisson process for the arriving students. Then, we assign the current simulation clock time, `TNOW`, to the attribute `myArrvalTime`. `TNOW` is a global, non-user assignable variable that contains the current simulation time represented in the base time units of the simulation. After getting marked with their

3. Statistical Analysis for Finite Horizon Simulation Models

arrival time, we increment the number of students attending the mixer. Then, the student entity performs the name tag activity for the designated time. Because there is a 50-50 chance associated with wandering or not wandering, we use a DECIDE with chance construct and set the type of the student via the `myType` attribute before sending the student on the rest of their path. Notice the use of labels to denote where to send the entity (student). If the student is a wanderer, there is a delay to indicate the time spent wandering, and then another DECIDE with chance to determine if the student timid or decides to visit the recruiting stations. At the recruiting stations, we see the common SEIZE-DELAY-RELEASE pattern for utilizing the recruiting resources for each of the two companies. After visiting the recruiters, the students exit the mixer. A label is used as a location for exiting the mixer, decrementing the number of students in the system and recording the statistics by type of student. Finally, the student entities are disposed.

3.4.2. Implementing the Model

Now, we are ready to translate the conceptual model into the simulation model by using the Arena environment. The completed model can be found in the book support files for this chapter entitled, *STEM_Mixer.doe*. The pseudo-code should be used to guide the model building process. The following Arena modules will be used in the implementation.

- ATTRIBUTE – to define the `myType`, `myArrivalTime`, and `mySysTime` attributes
- VARIABLE – to define the variable, `vNumInSys`
- RESOURCE – to define the MalWart and JHBunt resources and their capacities
- TIME PERSISTENT – to define the time-persistent statistical collection on the variable tracking the number of students attending the mixer
- CREATE – to specify the arrival pattern of the students
- ASSIGN – as per the pseudo-code to set the value of attributes or variables within the model a particular instances of time
- PROCESS – for delaying for the name tags, wandering around, and talking to the recruiters
- DECIDE – to decide whether to wander or not, whether to leave without visiting the recruiters, and to collect statistics by type of student
- Goto Label – to direct the flow of entities (students) to appropriate model logic
- Label – to define the start of specific model logic
- RECORD – collect statistics on observation-based (tally) data, specifically the system time
- DISPOSE – to define the end of the processing of the students and release the memory associated with the entities

Except for the TIME PERSISTENT module, which is found on the Statistics Panel, all of the other modules are found on the Basic Process Panel. The first step in the model implementation process is to access the ATTRIBUTE, VARIABLE, RESOURCE, and TIME PERSISTENT modules. These modules are all called data modules. They have a tabular layout that facilitates the addition of rows of information. By right-clicking on a row, you can also view the dialog box associated with the module. The “spreadsheet” view of the data modules is shown in Figures 3.6, 3.7, 3.8, and 3.9.

Attribute - Basic Process						
	Name	Comment	Rows	Columns	Data Type	Initial Values
1 ►	myArrivalTime				Real	0 rows
2	myType				Real	0 rows
3	mySysTime				Real	0 rows

Double-click here to add a new row.

Figure 3.6.: Attribute data module

Variable - Basic Process							
	Name	Comment	Rows	Columns	Data Type	Clear Option	File Name
1 ►	vNumInSys				Real	System	0 rows <input checked="" type="checkbox"/>

Double-click here to add a new row.

Figure 3.7.: Variable data module

Resource - Basic Process								
	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures
1	nhlWarrRecruiters	Fixed Capacity	2	0.0	0.0	0.0		0 rows <input checked="" type="checkbox"/>
2 ►	rjhBunRecruiters	Fixed Capacity	3	0.0	0.0	0.0		0 rows <input checked="" type="checkbox"/>

Double-click here to add a new row.

Figure 3.8.: Resource data module

An example of the Edit by Dialog view is shown in Figure 3.10 for the TIME PERSISTENT module. Notice that every Edit by Dialog view has a Help button that allows easy access to the Arena Help associated with the module. The Arena Help will include information about the text field prompts, the options associated with the module as well as examples and their meaning. The TIME PERSISTENT module requires a name for the statistic being defined as well as an expression that will be observed for statistical collect over time. In this case, the variable, myNumIn-Sys, is used as the expression.

Figure 3.11 shows the module layout for creating the students and getting them flowing into the mixer. These modules follow the pseudo-code very closely. Notice in Figure 3.12 the edit by dialog view of the CREATE module is provided. Since we have a Poisson process with rate 0.5 students per minute, the CREATE module is completed in a similar fashion as we discussed in Chapter 2 by specifying the *time between arrival* process.

3. Statistical Analysis for Finite Horizon Simulation Models

Time Persistent - Statistics				
	Name/Report Label	Expression	Collection Period	Output File
1 ►	Avg Num In System	vNumInSys	Entire Replication	
Double-click here to add a new row.				

Figure 3.9.: Time persistent data module

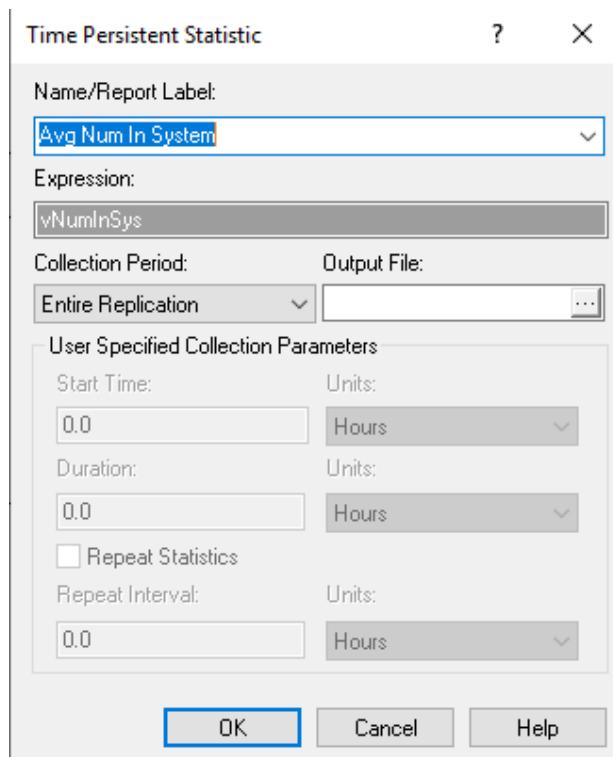


Figure 3.10.: Time persistent dialog module view

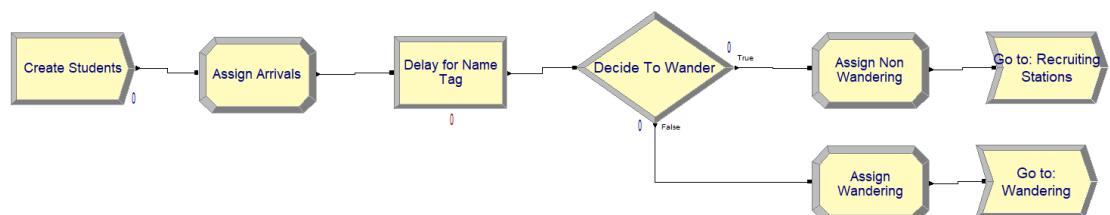


Figure 3.11.: Creating students for STEM Career Mixer

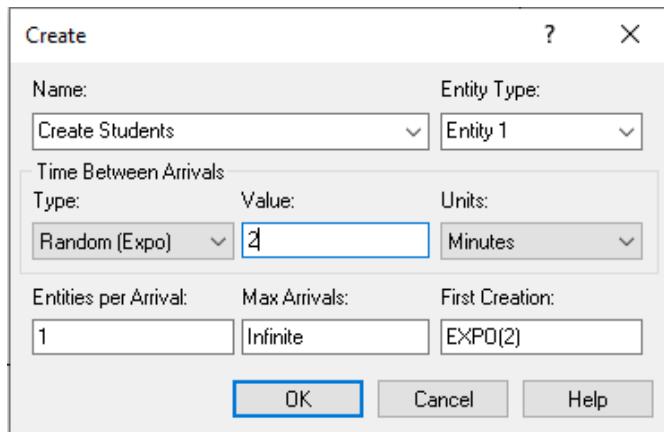


Figure 3.12.: CREATE module for STEM Career Mixer

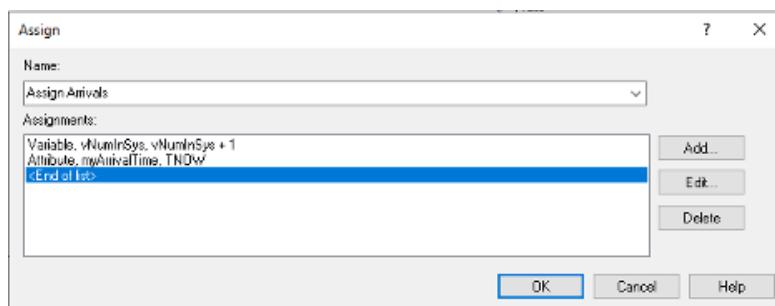


Figure 3.13.: ASSIGN arrivals module

 **Why are we not using the Poisson distribution in the CREATE module?** This is a common point of confusion. The CREATE module requires the **time between events**, while a Poisson distribution models the *number* of events. While it is theoretically possible that the Poisson distribution could be used to represent the time between events, it would indeed be a rare occurrence if it was. If you *ever* want to use the Poisson distribution in your CREATE module then you are either wrong in your thinking or you better know exactly what you are doing! The Poisson distribution models a count of the number of events in a time period. The random variable, time between arrivals, represents time. As such, it is a continuous random variable and thus the Poisson distribution (being discrete) would be highly inappropriate. The confusion stems from the wording of the problem, which commonly states that we have a Poisson process with mean rate λ for the arrivals. There is a theorem in probability that states that if we have a Poisson process with mean rate λ , then the time between events is governed by an exponential distribution with a mean of $\theta = 1/\lambda$. Thus, we would use the Random(expo) option in the CREATE module with the value of θ specified for the mean of the distribution.

3. Statistical Analysis for Finite Horizon Simulation Models

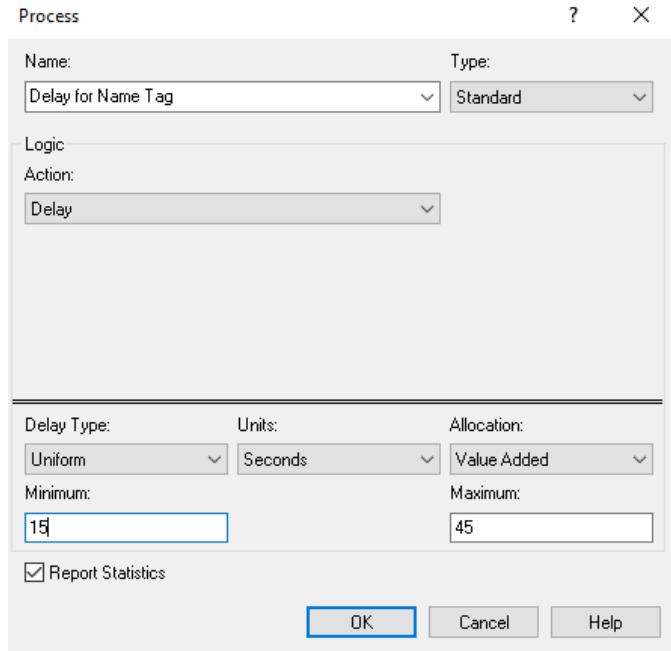


Figure 3.14.: DELAY for name tags

Next, Figure 3.13 illustrates the first ASSIGN module, which increments the number of students in the system and assigns the arrival time of the student. Figure 3.14 shows the dialog box for a PROCESS module with the Delay action to implement the delay for putting on the name tag. Notice that this option does not require a resource as we saw in the Chapter 2. We simply need to specify the Delay action and type of the delay with the appropriate distribution (UNIF(15, 45)), in seconds.

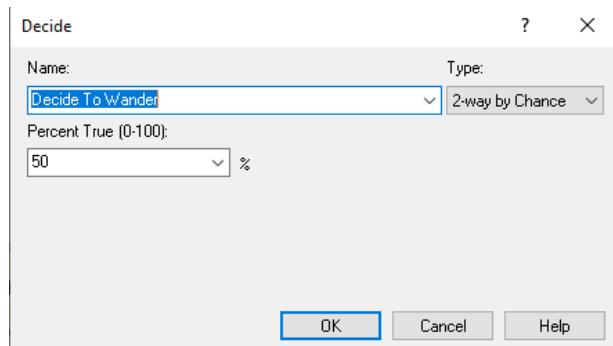


Figure 3.15.: DECIDE to wander or not

Continuing along with the modules in Figure 3.11, the DECIDE module (shown in Figure 3.15) is used with the 2-way by Chance option, where the branching chance is set at 50%. The two ASSIGN modules simply assign myType to the value 1 (non-wanders) or 2 (wanderers) depend-

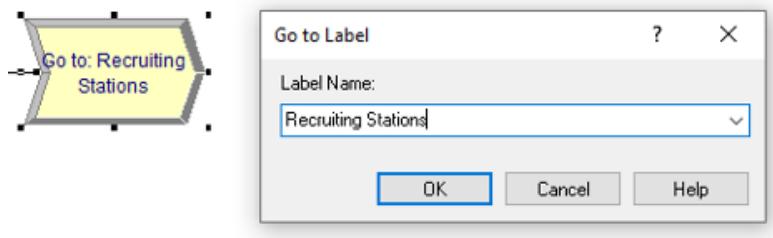


Figure 3.16.: GOTO Label: Recruiting Stations

ing on the result of the random path assignment from the DECIDE with chance option. Finally, we see in Figure 3.16, the GO TO LABEL module, which looks a little like an arrow and simply contains the specified label to where the entity will be directed. In this case, the label is the “Recruiting Stations”.

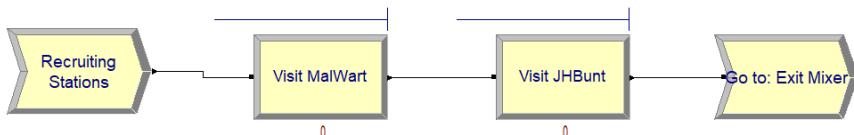


Figure 3.17.: Recruiting station process logic

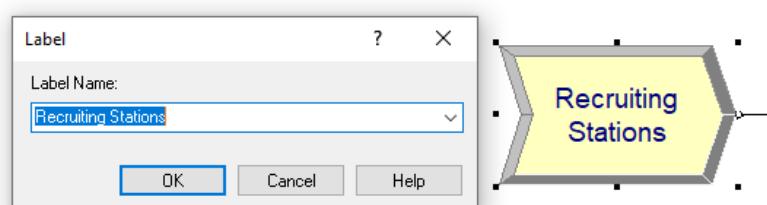


Figure 3.18.: Recruiting station label

Figure 3.17 shows the module layout for the recruiting station logic. Notice that Figure 3.18 shows the specification of the label marking the start of the logic. Figure 3.19 show the PROCESS module for the students visiting the MalWart recruiting station. Similar to its use in Chapter 2, this module uses the SEIZE-DELAY-RELEASE option, specifies that the rMalWartRecruiter resource will have one of its two recruiters seized and provides the time delay, EXPO(3) minutes, for the delay time.

Figure 3.20 illustrates the module flow logic for the students that wander. Here, we again use a PROCESS module with the DELAY option, a DECIDE with 2-way by Chance option and GO TO LABEL modules to direct the students. The ASSIGN module in Figure 3.20 simply sets the attribute, myType, to the value 3, as noted in the pseudo-code.

Finally, Figure 3.21 presents the model flow logic for collecting statistics when the students leave

3. Statistical Analysis for Finite Horizon Simulation Models

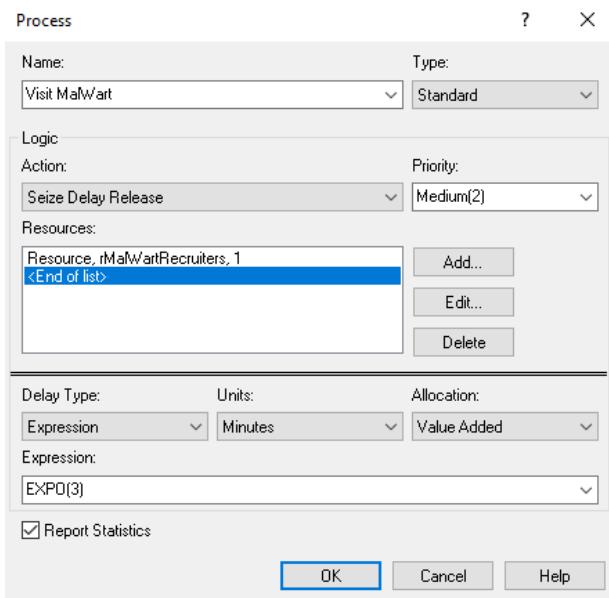


Figure 3.19.: Visting MalWart PROCESS module

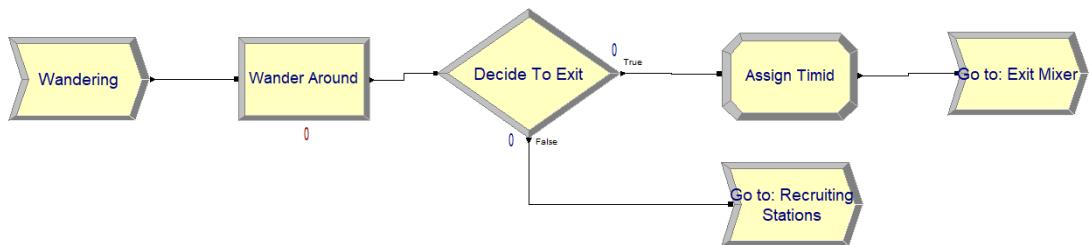


Figure 3.20.: Wandering students process logic

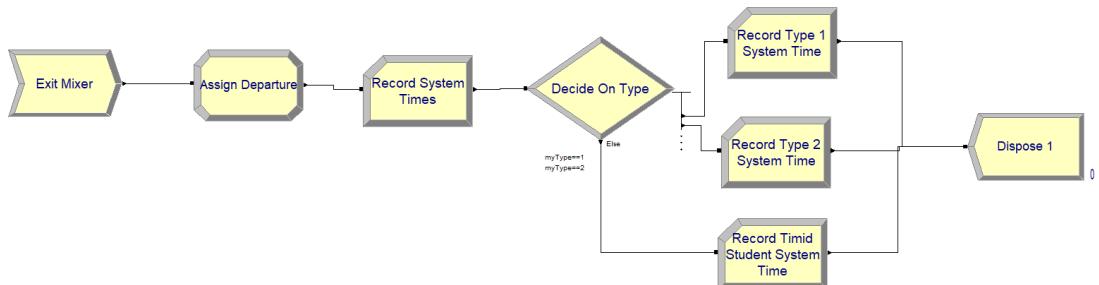


Figure 3.21.: Statistics collection exiting logic

the mixer. The ASSIGN module of Figure 3.21 is shown in Figure 3.22.

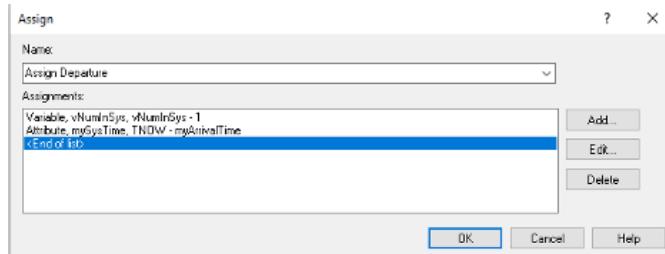


Figure 3.22.: ASSIGN module for departures

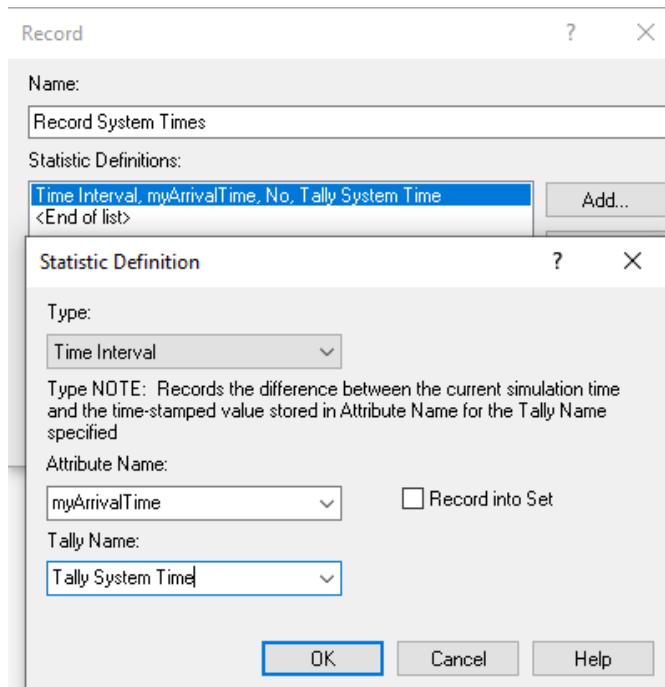


Figure 3.23.: RECORD module for system time interval option

In the ASSIGN module, first the number of students in attending the mixer is decremented by 1 and the value of the attribute, mySysTime, is computed. This holds the value of TNOW – myArriveTime, which implements $T_i = D_i - A_i$ for each departing student. Figure 3.23 shows the Time Interval option of the RECORD module which, as noted in the dialog box, "Records the difference between the current simulation time and the time-stamped value stored in the Attribute Name for the Tally Name specified." This is, TNOW – myArriveTime, which is the system time.

Since this type of recording is so common the RECORD module provides this option to facilitate this type of collection. The RECORD module has five different record types:

Count When Count is chosen, the module defines a COUNTER variable. This variable will be

3. Statistical Analysis for Finite Horizon Simulation Models

incremented or decremented by the value specified.

Entity Statistics This option will cause entity statistics to be recorded.

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.

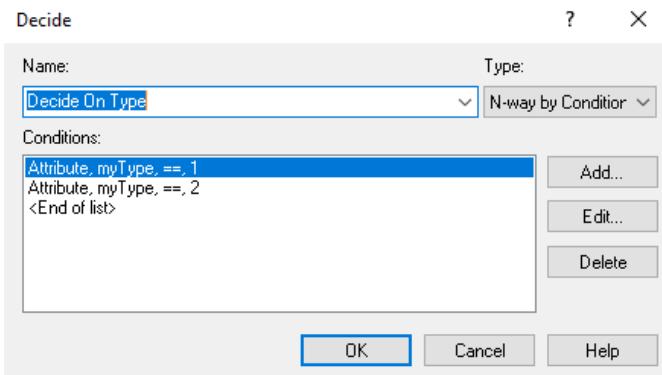


Figure 3.24.: DECIDE module for checking type of student

Figure 3.24 shows a DECIDE module with the N-way by Condition option. This causes the DECIDE module to have more than two output path ports. The order of the listed conditions within the module correspond to the output ports on the module. Thus, entities that come out of the first output port have the value of attribute myType equal to 1. The entities entering the DECIDE module must choose one of the paths. Figure 3.25 shows the RECORD module for collection of system time statistics on the entities that have the myType attribute equal to 1. Notice that I have elected to use the Expression option for defining the statistical collection and specified the value of the attribute, mySysTime, as the recorded value. This is the same as choosing the Time Interval option and choosing the myArrivalTime attribute as the time-stamped attribute. This is just another way to accomplish the same task. In fact, the Time Interval option of the RECORD module was invented because this is such a very common modeling task. The rest of the modules in Figure 3.21 are completed in a similar fashion.

3.4.3. Planning the Sample Size

Now we are ready to run the model and see the results. Figure 3.26 shows the Run Setup configuration to run 5 replications of 6 hours with the base time units in minutes. This means that

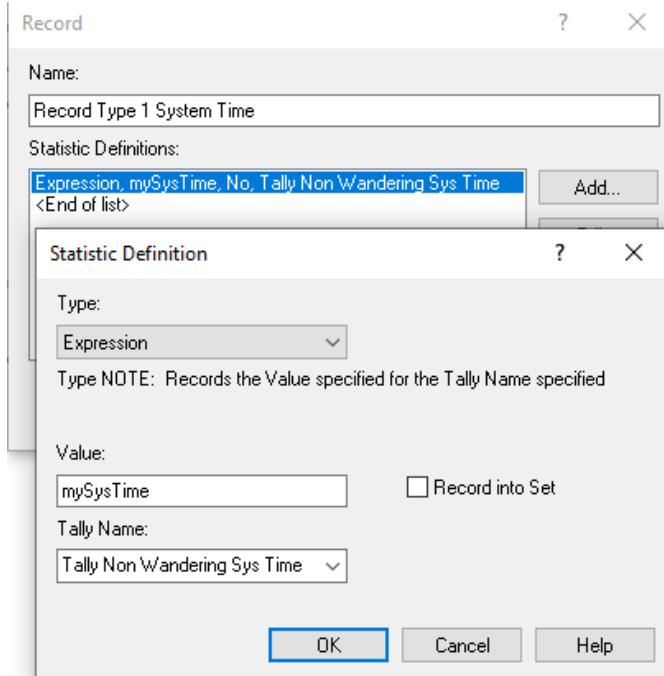


Figure 3.25.: Recording statistics on type 1 students

the simulation will repeat the operation of the STEM mixer 5 times. Each replication will be independent of the others. Thus, we will get a random sample of the operation of 5 STEM mixers over which we can report statistics.

Running the simulation results in the output for the user defined statistics as shown in Figure 3.27. We see that the average system time is 39.192 minutes with a half-width of 9.09 with 95% confidence. Notice also that in Figure 3.27 there are two time-persistent statistics reported for the number of students in the system, both reporting the same statistics. The first one listed is due to our use of the TIME-PERSISTENT module. The second one listed is because, in the VARIABLE module, the report statistic option was checked for the variable vNumInSys. These two approaches result in the same results so using either method is fine. We will see later in the text that using the TIME PERSISTENT module will allow the user to save the collected data to a file and also provides options for collecting statistics over user defined periods of time.

Since the STEM mixer organizer wants to estimate the average time students spend at the mixer, regardless of what they do, to be within plus or minus 1 minute with a 95% confidence level, we need to increase the number of replications. To determine the number of replications to use we can use the methods of Section 3.4.1. Noting the initial half-width of $h_0 = 9.09$ for $n_0 = 5$ replications, the desired half-width, $h = 1$ minute, and applying Equation (3.12), we have:

$$n \cong n_0 \left(\frac{h_0}{h} \right)^2 = 5 \left(\frac{9.09}{1} \right)^2 = 413.14 \approx 414$$

3. Statistical Analysis for Finite Horizon Simulation Models

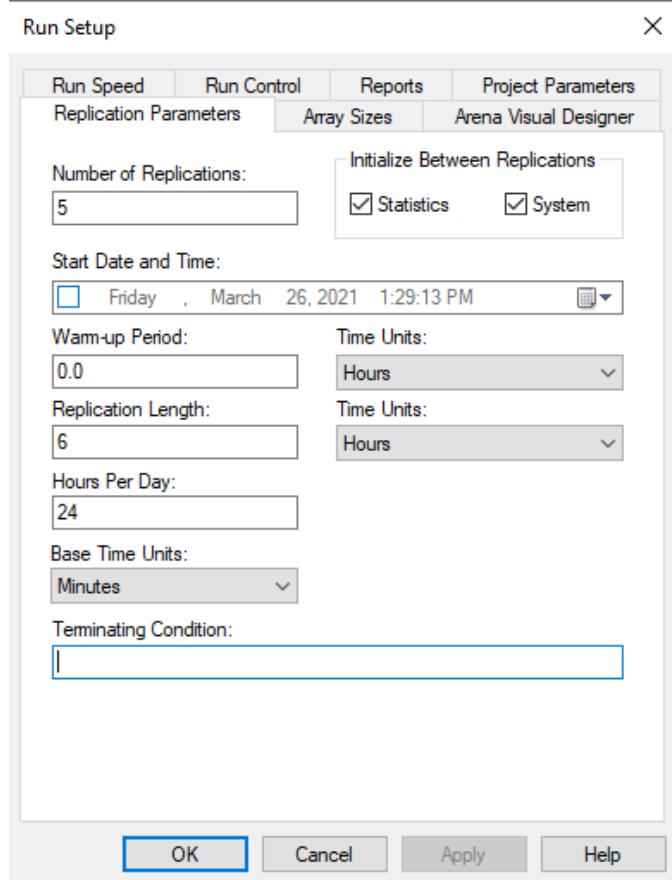


Figure 3.26.: Run setup dialog for STEM Mixer example

Tally						
Expression	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Tally Non Wandering Sys Time	26.9080	7.45	22.2295	35.7923	1.5398	72.6515
Tally Timid Student Sys Time	28.4029	3.01	25.9575	31.3376	18.2033	43.9091
Tally Wandering Sys Time	55.5002	9.36	47.9587	66.0575	17.5103	108.20
Interval	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Tally System Time	39.1920	9.09	32.9411	49.1896	1.5398	108.20

Time Persistent						
Time Persistent	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Avg Num In System	19.3403	3.36	16.7246	23.2630	0.00	39.0000
Variable	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
vNumInSys	19.3403	3.36	16.7246	23.2630	0.00	39.0000

Figure 3.27.: STEM Mixer simulation results for 5 replications

Updating the number of replications to run to be 414 and executing the model again yields the results in Figure 3.28. We see from Figure 3.28 that the desired half-width criterion of less than 1 minute is met, noting that the resulting half-width is less than 0.79 minutes.

Tally

Expression	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Tally Non Wandering Sys Time	21.8025	< 0.79	10.1882	59.8097	0.4445	104.65
Tally Timid Student Sys Time	26.9312	< 0.23	19.9311	34.7120	15.5254	44.9705
Tally Wandering Sys Time	49.4055	< 0.87	36.3332	96.6045	17.4824	141.35
Interval	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Tally System Time	33.9325	< 0.79	21.8254	76.5154	0.4445	141.35

Time Persistent

Time Persistent	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Avg Num In System	16.6950	< 0.45	9.2962	40.5220	0.00	58.0000
Variable	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
vNumInSys	16.6950	< 0.45	9.2962	40.5220	0.00	58.0000

Figure 3.28.: STEM Mixer simulation results for 414 replications

To use the Normal approximation method of determining the sample size based on the Arena output, we must first compute the sample standard deviation for the initial pilot sample and then use Equation (3.6):

$$n \geq \left(\frac{z_{1-(\alpha/2)} s}{\epsilon} \right)^2$$

To compute the sample standard deviation from the half-width, we have:

$$s_0 = \frac{h_0 \sqrt{n_0}}{t_{1-(\alpha/2), n-1}} = \frac{9.09 \times \sqrt{5}}{t_{0.975, 4}} = \frac{9.09 \times \sqrt{5}}{2.7764} = 7.3208$$

Then, the normal approximation yields:

$$n \geq \left(\frac{z_{1-(\alpha/2)} s}{\epsilon} \right)^2 = \left(\frac{1.9599 \times 7.3208}{1} \right)^2 = 205.8807 \cong 206$$

We can also use an iterative approach based on Equation (3.5).

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

3. Statistical Analysis for Finite Horizon Simulation Models

Notice that the equation emphasizes the fact that the half-width, $h(n)$ and the sample standard deviation, $s(n)$, both depend on n in the formula. Using goal-seek within the spreadsheet that accompanies this chapter yields $n \geq 208.02 = 209$ as illustrated in Figure 3.29.

A	B	C	D	E
1				
2	alpha =	0.05		
3	Initial std dev =	7.3208		
4	Desired half-width =	1		
5	Recommended n =	208.0214		
6	alpha/2 =	0.025		
7	t_alpha/2, n-1	1.97149		
8	half-width	1.000689		
9	difference	0.000689		
10				
11				
12				
13				
14				

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 3.29.: Goal Seek Results for Student-t Iterative Method

To summarize, we have the following recommendations for the sample size based on the three methods.

Table 3.6.: Sample size recommendation based on initial sample size, n = 5

Method	Recommended Sample Size	Resulting Half-width
Half-Width Ratio	414	< 0.79
Normal Approximation	206	< 1.16
Iterative Student T Method	209	< 1.18

As you can see from the summary in Table 3.6, the methods all result in different recommendations and different resulting half-widths. Why are the recommendations different? Because each method makes different assumptions. In addition, you must remember that the results shown here are based on an initial pilot run of 5 samples. Using a larger initial pilot sample size will result in different recommendations. Running the model for 10 replications yields an initial half-width of 5.4 minutes and the recommendations in Table 3.7.

Table 3.7.: Sample size recommendation based on initial sample size, n = 10

Method	Recommended Sample Size	Resulting Half-width
Half-Width Ratio	292	< 0.96
Normal Approximation	219	< 1.14
Iterative Student T Method	221	< 1.14

3.5. Using Sequential Sampling Methods on a Finite Horizon Simulation

Based on these results, we can be pretty certain that the number of replications needed to get close to 1 minute half-width with 95% confidence is somewhere between 219 and 292. So, which method should you use? Generally, the half-width ratio method is the most conservative of the approaches and will recommend a larger sample size.

Why does determining the sample size matter? This model takes less than a second per replication. But what if a single replication required 1 hour of execution time. Then, we will really care about determining to the extent possible the minimum number of samples needed to provide a result upon which we can make a confident decision. Thus, there is a trade-off between the time needed to get a result and the desired precision of the result. This trade-off motivates the use of sequentially sampling until the desired half-width is met. This is the topic of the next section.

3.5. Using Sequential Sampling Methods on a Finite Horizon Simulation

In this section, we will discuss how to determine the sample size for a finite horizon simulation by using a sequential sampling method. The new concepts introduced within this section include:

- Defining an OUTPUT statistic using the Statistics Panel
- Using a “logical entity” within a model
- The MREP and NREP global variables

The methods discussed for determining the sample size are based on pre-determining a *fixed* sample size and then making the replications. If the half-width equation is considered as an iterative function of n .

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

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. For this method, the sample size of the experiment is not known in advance. That is, the sample size is actually a random variable. 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 automatically stop when the criterion is met.

The pseudo-code for this concept is provided here.

3. Statistical Analysis for Finite Horizon Simulation Models

```
CREATE a logical entity at time 0.0
DECIDE IF NREP <= 2 THEN
    DISPOSE
ELSE IF half-width <= error bound THEN
    ASSIGN MREP = NREP
END DECIDE
DISPOSE
```

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 the that maximum number of replications, MREP, has been reached by setting MREP equal to the current replication number, NREP.

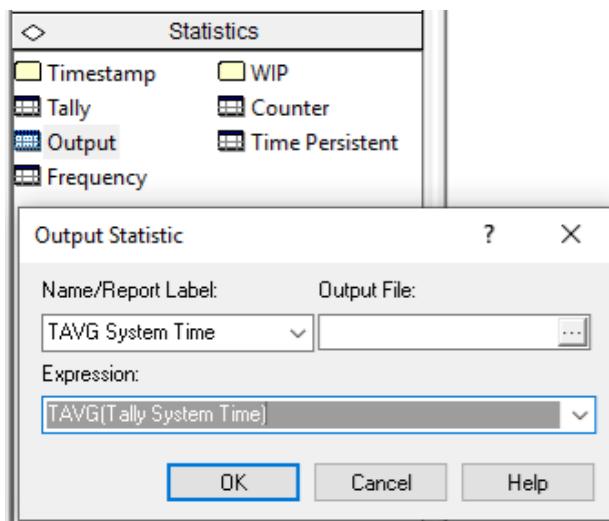


Figure 3.30.: Defining an OUTPUT statistic

In order to implement this pseudo-code, you must first define an OUTPUT statistic. This is done in Figure 3.30, 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. An OUTPUT statistic collects statistics on the expression indicated at the end of a replication. Thus, a single observation, one for each replication, is observed and standard sample statistics, such as the sample average, minimum, maximum, and

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.

Figure 3.31 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

3.5. Using Sequential Sampling Methods on a Finite Horizon Simulation

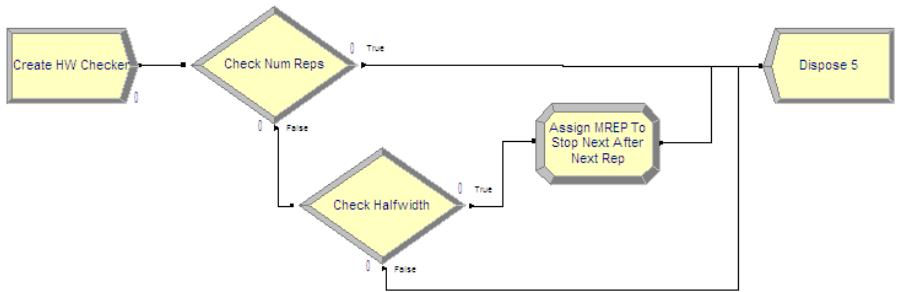


Figure 3.31.: Implemented half-width checking logic

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 3.32 shows that the second DECIDE module uses the function `ORUNHALF(Output ID)` to check the current half-width versus the error bound, which was 1 minute 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.

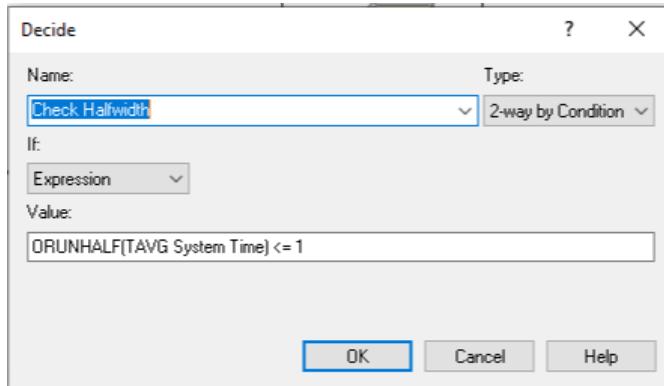


Figure 3.32.: Checking the ORUNHALF function

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

3. Statistical Analysis for Finite Horizon Simulation Models

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 3.33 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).

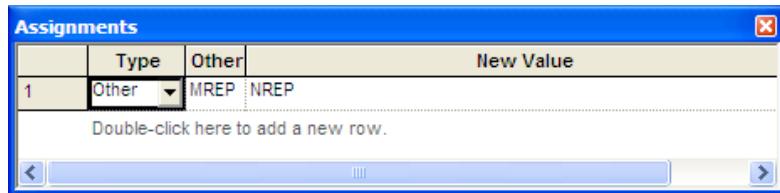


Figure 3.33.: Assigning MREP to NREP

This has been implemented in the file *STEM_Mixer_Sequential_Sampling.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 3.34.

Tally						
Expression	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Tally Non Wandering Sys Time	21.8187	< 1.00	10.9668	59.8097	0.4976	104.65
Tally Timid Student Sys Time	26.9227	< 0.29	19.9311	34.7120	15.5254	44.9705
Tally Wandering Sys Time	49.2774	< 1.10	36.3332	96.6045	17.4824	141.35
Interval	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Tally System Time	33.8558	< 0.99	21.8254	76.5154	0.4976	141.35
Time Persistent						
Time Persistent	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Avg Num In System	16.6623	< 0.57	9.4260	40.5220	0.00	58.0000
Variable	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
vNumInSys	16.6623	< 0.57	9.4260	40.5220	0.00	58.0000
Output						
Output	Average	Half Width	Minimum Average	Maximum Average		
TAVG System Time	33.8558	0.99	21.8254	76.5154		

Figure 3.34.: Sequential sampling results for N = 280 Replications

In the sequential sampling experiment there were 280 replications. We saw from Table 3.7 that the sample size should be between 219 and 292 in order to meet the half-width criteria based

on an initial sample size of 10 replications. The sequential sampling approach iterates to the minimum number of replications to meet the criteria.

3.6. 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 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, `STEM_Mixer_Frequencies.doe` that accompanies this chapter illustrates the use the frequencies option. Figure 3.35 and Figure 3.36 show how to collect frequencies on a queue and on a resource. In Figure 3.36, 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 frequencies option will collect the time spent in each of these categories and also the percentage of the total time within each category.

In Figure 3.36, the frequency option is used to automatically track the states of the JHBunt recruiter resource. Resources automatically have a default state set defined which include `BUSY` and `IDLE`. Chapter 6 discusses describes additional resource modeling that includes states `INACTIVE` and `FAILED`. Since the frequency option will collect the percentage of time spent in the associated states, this represents another method for collecting the utilization of resources. More

3. Statistical Analysis for Finite Horizon Simulation Models

Statistic - Advanced Process									
	Name	Type	Frequency Type	Expression	Collection Period	Report Label	Report Label	Categories	
1	JHBuntRFreq	Frequency	State	Expression 1	Entire Replication	JHBuntRFreq	JHBuntRFreq	0 rows	
2 ►	JHBuntQFreq	Frequency	Value	NQ(Visit JHBunt.Queue)	Entire Replication	JHBuntQFreq	JHBuntQFreq	3 rows	
3	JHBuntQProbEm					JHBuntQProbEmpty	JHBuntQProbEmpty	0 rows	

Double-click here to add a new row.

Categories					
	Constant or Range	Value	High Value	Category Name	Category Option
1	Constant	0		A_EmptyQ	Include
2	Range	0	5	B_OneToFive	Include
3	Range	5	10	C_SixToTen	Include

Double-click here to add a new row.

Figure 3.35.: Setting up the frequency option for the JHBunt queue using the statistics module

generally, you can define any state set and track statistics on time spent in the states based on general model logic.

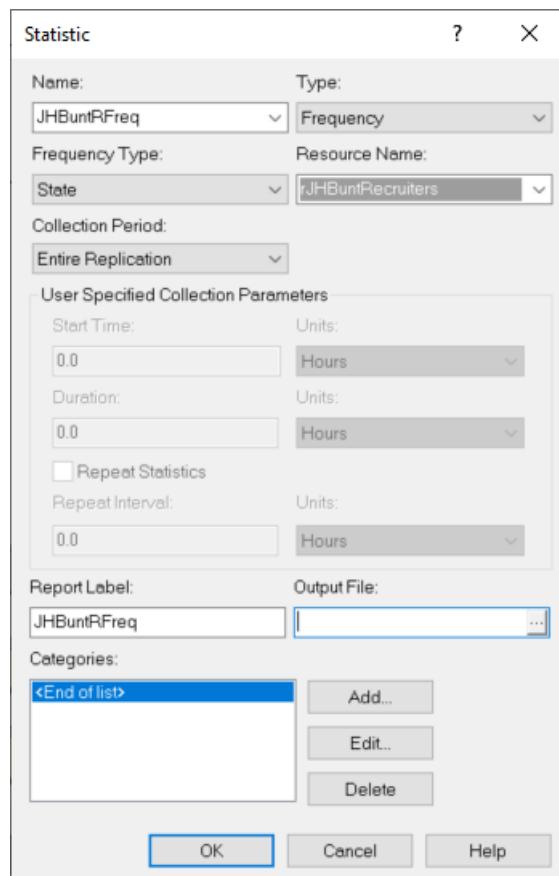


Figure 3.36.: Setting up the frequency option for the JHBunt resource using the statistics module

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.

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 percent 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 percent 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 non-excluded categories (i.e., categories that have a value in the restricted percent 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 3.37 illustrates the results from running the STEM mixer model for 30 replications of 360 minutes. For the frequency tabulation on the number students in the JHBunt recruiter queue,

3. Statistical Analysis for Finite Horizon Simulation Models

Replication 1		Start Time:	0.00	Stop Time:	360.00	Time Units:	Minutes
JHBuntQFreq		Number Obs	Average Time	Standard Percent	Restricted Percent		
A_EmptyQ		16	7.8216	34.76		37.36	
B_OneToFive		23	6.1585	39.35		42.29	
C_SixToTen		14	4.8683	18.93		20.35	
OUT OF RANGE		6	4.1754	6.96		--	
JHBuntRFreq		Number Obs	Average Time	Standard Percent	Restricted Percent		
BUSY		5	70.0979	97.36		97.36	
IDLE		5	1.9021	2.64		2.64	

Figure 3.37.: Replication 1 frequency output for STEM Mixer

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 in the figure, the OUT OF RANGE category is automatically excluded from the restricted percentage column. If you add up the percentages in the standard percent column ($34.76 + 39.35 + 18.93 + 6.96 = 100$), we see that they add up to 100 percent. The restricted percent column excludes the OUT OF RANGE category and redistributes the time accordingly. Figure 3.38 illustrates the spreadsheet tabulation of the frequency results. This file is available in the book support files for this chapter as *FrequenciesCalculations.xlsx*. If the number of observations is multiplied by the average time in the category, then we get the total time spent in the category. The total of this time should be equal to the time over which the category was observed, in this case 360 minutes. Notice that the OUT OF RANGE category is restricted from the total time to produce the restricted percentages.

	A	B	C	D	E
1	Number Obs	Average Time	Total Time	Standard Percent	Restricted Percent
2	16	7.8216	125.1456	34.76%	37.36%
3	23	6.1585	141.6455	39.35%	42.29%
4	14	4.8683	68.1562	18.93%	20.35%
5	6	4.1754	25.0524	6.96%	
6			360.0		
7					

	A	B	C	D	E
1	Number Obs	Average Time	Total Time	Standard Percent	Restricted Percent
2	16	7.8216	=A2*B2	=C2/\$C\$6	=C2/(\$C\$6-\$C\$5)
3	23	6.1585	=A3*B3	=C3/\$C\$6	=C3/(\$C\$6-\$C\$5)
4	14	4.8683	=A4*B4	=C4/\$C\$6	=C4/(\$C\$6-\$C\$5)
5	6	4.1754	=A5*B5	=C5/\$C\$6	
6			=SUM(C2:C5)		

Figure 3.38.: Spreadsheet tabulation of frequency data

When using frequencies, the frequencies will be tabulated for each replication; however, frequency statistics are not automatically tabulated across replications. To tabulate across repli-

3.6. Tabulating Frequencies using the STATISTIC Module

cations, you can define an OUTPUT statistic and use one of the previously discussed frequency functions (e.g. FRQTIM, FAVG, etc.). 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. Figure 3.39 illustrates how to set up an OUTPUT statistic to collect the percentage of time that the queue is empty across the replications.

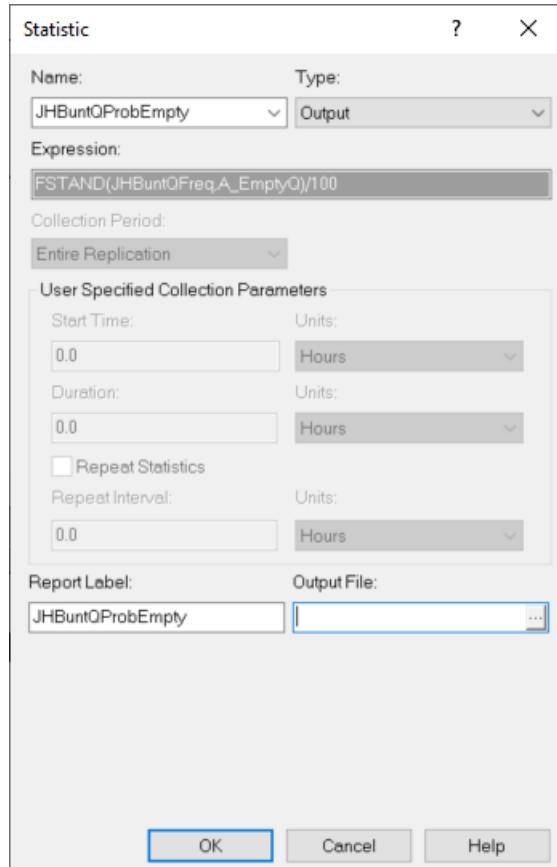


Figure 3.39.: Collecting frequency statistics across replications

The results from running the model for 30 replications are shown in Figure 3.40. We see that there is about a 38% of the time the queue is empty.

In the following section, we summarize the coverage of this chapter.

3. Statistical Analysis for Finite Horizon Simulation Models

Tally

Expression	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Tally Non Wandering Sys Time	21.6787	3.17	11.5121	45.3989	0.6570	84.0867
Tally Timid Student Sys Time	26.5688	0.87	21.4979	31.3376	15.8914	43.9091
Tally Wandering Sys Time	48.5243	3.14	37.9558	71.8209	17.5103	113.90
Interval	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Tally System Time	33.5095	3.14	22.8075	58.0200	0.6570	113.90

Time Persistent

Time Persistent	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Avg Num In System	16.6423	1.92	10.3994	32.5527	0.00	55.0000
Variable	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
vNumInSys	16.6423	1.92	10.3994	32.5527	0.00	55.0000

Output

Output	Average	Half Width	Minimum Average	Maximum Average
JHBuntQProbEmpty	0.3807	0.06	0.05712865	0.6984

Figure 3.40.: Across replication results for probability that the queue is empty

3.7. Summary

This chapter described many of the statistical aspects of simulation that will be typically encountered when performing a simulation study. An important aspect of performing a correct simulation analysis is to understand the type of data associated with the performance measures (time-persistent versus tally-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 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. These topics will be covered in Chapter 5.

This chapter also provided additional examples of the following Arena modules:

- CREATE
- ASSIGN
- PROCESS
 - SEIZE, DELAY, RELEASE option

- DELAY option
- DECIDE
 - 2 way with condition
 - N-way with condition
 - 2 way with chance
- GOTO LABEL
- LABEL
- RECORD
 - expression option
 - time interval option
- Statistical collection
 - OUTPUT
 - TIME PERSISTENT
 - TALLY
 - FREQUENCIES

A RECORD module is placed within the model window. It is used to record observations at particular event times as denote by where the module is placed within the flow path of the entity within the model. The RECORD module using the “expression” option will compute statistics on the value computed by the expression. The critical thing to remember about the RECORD module is that it is executed when an entity passes through it. Thus, what it observes (records) will be dependent upon where it is placed and when it is executed.

The STATISTIC module is used to define and collect additional statistics that occur during a simulation. The STATISTIC module also allows for the definition of files to store observed values. First, it allows the modeler to pre-define Tallies and Counters, which are what RECORD modules use. This is similar to using the RESOURCE module to define resources that are used in the model window, or pre-defining variables or attributes before using them in the model. Finally, the STATISTIC module allows the user to define OUTPUT statistics. These work in a similar fashion as how RECORD works, except the expression is *observed one time, at the end of the replication*. While a RECORD module can observe values at any simulation time, the OUTPUT statistic option of the STATISTIC module can only observe values at the end of a replication.

The STATISTC module also allows for the definition of Time Persistent statistical collection on variables defined in the model. This functionality is the same as checking the “collect statistics” box when defining a variable; however, it also provides additional functionality. First, the modeler can provide their own name for the time persistent statistic and may choose to save the data into a file. Secondly, through the STATISTIC module, the modeler can define periods of time over which statistics can be collected on a variable. This is very useful in simulation situations that have time-varying inputs. Finally, the STATISTIC module allows for the definition of FREQUENCIES to provide finer detailed collect of time persisting variables in states or categories.

3. Statistical Analysis for Finite Horizon Simulation Models

This chapter has covered almost all of the most useful aspects of collecting statistics from your simulation model. As a final note, there are other statistics that are automatically collected by Arena, some of which do not add much value to the analysis. For example, when you run your model and view the category reports, the very first item that you see is summary about the number of entities in and out of the model. I strongly recommend **ignoring** these numbers. They are pretty much useless unless you commit to using Entity Types, for which I have never found a strong use case, except for applications involving activity based costing (See Appendix D).

Now that you have a solid understanding of how to program and model in Arena and how to analyze your results, you are ready to explore the application of simulation to process modeling situations involving more complex systems. These systems form the building blocks for modeling systems in manufacturing, transportation, and service industries.

3.8. Exercises

Exercise 3.1. 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.

Exercise 3.2. Provide the missing information for steps for the modeling questions.

- What is the _____?
 - What are the elements of the _____?
 - What information is known by the _____? - What are the required performance _____?
 - What are the _____ types?
 - What information must be recorded or remembered for each _____ instance?
 - How are entities (_____ instances) introduced into the system?
 - What are the _____ that are used by the entity types?
 - Which entity types use which _____ and how?
 - What are the process flows? Sketch the process or make an _____
 - Develop _____ for the situation
 - Implement the model in Arena
-

Exercise 3.3. The method of _____ assumes that a random sample is formed when repeatedly running the simulation.

Exercise 3.4. Indicate whether or not the statistical quantity should be classified as tally-based or time-persistent.

3. Statistical Analysis for Finite Horizon Simulation Models

Classification	Description
	The number of jobs waiting to be processed by a machine
	The number of jobs completed during a week
	The number in customers in queue:
	The time that the resource spends serving a customer
	The number of items in sitting on a shelf waiting to be sold
	The waiting time in a queue
	The number of patients processed during the first hour of the day
	The time that it takes a bus to complete its entire route
	The number of passengers departing the bus at Dickson street
	The amount of miles that a truck travel empty during a week

Exercise 3.5. What module is used to compute statistical results across replications?

Exercise 3.6. *True or False* A replication is the generation of one sample path which represents the evolution of the system from its initial conditions to its ending conditions.

Exercise 3.7. In a _____ horizon simulation, a well defined ending time or ending condition can be specified which clearly demarks the end of the simulation.

Exercise 3.8. *True or False* We use the Arena function, TAVG(), for writing out statistics on the average of observation-based variables.

Exercise 3.9. Which of the following are finite horizon situations? Select all that apply.

- a. Bank: bank doors open at 9 am and close at 5 pm
- b. Military battle: simulate until force strength reaches a critical value
- c. A factory where we are interested in measuring the steady state throughput
- d. A hospital emergency room which is open 24 hours a day, 7 days a week

Exercise 3.10. 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 a part. Given the half-width of the system time after 10 runs is 47.25, compute the sample size using the half-width ratio method. Round the answer to the next highest integer.

Exercise 3.11. 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 a part. Given the half-width of the system time after 10 runs is 47.25. Find the approximate number of replications needed in order to have a 99% confidence interval that is within plus or minus 2 minutes of the true mean system time using the half-width ratio method.

Exercise 3.12. Assume that the following results represent the summary statistics for a pilot run of 10 replications from a simulation for the system time in minutes. $\bar{x} = 78.2658 \pm 9.39$ with confidence 95%. Find the approximate number of *additional* replications in order to have a 99% confidence interval that is within plus or minus 2 minutes of the true mean system time.

Exercise 3.13. 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].

- a. Find the approximate number of samples needed to have a 95% confidence interval that is within plus or minus 2 minutes of the true mean system time.
 - b. Find the approximate number of samples needed to have a 99% confidence interval that is within plus or minus 1 minute of the true mean system time.
-

Exercise 3.14. 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 half-width ratio sample size determination techniques, recommend a sample size to be 95% confident that you are within ± 0.10 of the true mean waiting time in the queue. The half-width ratio method requires a sample size of what amount?

Exercise 3.15. Assume that the following table represents the summary statistics for a pilot run of 10 replications from a simulation.

Simulation Statistics	NPV	P(NPV < 0)
Sample Average	83.54	0.4
Standard Deviation	39.6	0.15
Count	10	10

- a. Find the approximate number of additional replications to execute in order to have a 99% confidence interval that is within plus or minus 20 dollars of the true mean net present value using the normal approximation method.
 - b. Find the number of replications necessary to be 99% confident that you have an interval within plus or minus 2% of the true probability of negative present value.
-

Exercise 3.16. 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 inter-arrival 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 one-minute setup operation. The remaining 40 percent 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. The operator of the machine is required to both setup and operate the machine.

Run your model for 20000 minutes, with 10 replications. Report the utilization of the machines and operators. In addition, report the total time spent in the system for each type of part.

Exercise 3.17. 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 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 minutes to 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 a 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.

Exercise 3.18. Parts arrive to a small manufacturing cell at a rate of one part every 30 ± 20 seconds. Forty percent of the parts go to drill press, where one worker drills the hole in 60 ± 30 seconds. The rest of the parts go to the hole punch where one worker punches the hole in 45 ± 30 seconds. All parts must go to a deburring station, which takes 25 ± 10 seconds with one operator. For all parts, they then go through a heat treatment operation for 20 ± 10 minutes. For the purposes of this problem, there is always room within the heat treatment area to hold as many parts as necessary. The notation $X \pm Y$ indicates that the random variable is uniformly distributed over the range $(X-Y, X+Y)$. Build a model that can estimate the following performance measures: 1) average number of parts in the manufacturing cell, and 2) average time spent in the manufacturing cell. Run your model for 20 replications of 100 parts.

Exercise 3.19. Referring to Exercise 3.16. Determine the number of replications to ensure that you are 95% confident that the true expected system time for part type 2 has a half-width of 0.5 minutes or less.

Exercise 3.20. Referring to Exercise 3.17. Determine the number of replications to ensure that you are 95% confident that the true waiting time for the accountant is within ± 2 minutes.

Exercise 3.21. Referring to Exercise 3.18. Determine the number of replications to ensure that you are 95% confident that the average number of parts within the system is within ± 1 units.

3. Statistical Analysis for Finite Horizon Simulation Models

Exercise 3.22. Referring to Exercise 2.20. Determine the sample size necessary to estimate the mean shipment time for the truck only combination to within 0.5 hours with 95% confidence

Exercise 3.23. Referring to Exercise 2.22. Develop an model to estimate the average profit with 95% confidence to within plus or minus \$0.5.

4. Modeling Systems with Processes and Basic Entity Flow

LEARNING OBJECTIVES

By the end of this chapter, students should be able to:

- Gain a deeper understanding of the application of attributes and variables within models
- Understand what expressions are and how they can be applied within models
- Understand when ROUTE, STATION, and SEQUENCE modules can be used and apply them effectively for simple network modeling situations
- To be able to understand and model shared resources
- To be able to understand and model the duplication and batching of entities

Recall from Chapter 2 that a process is a sequence of activities experienced by entities as they move through a system. Activities represent tasks that take time and may require resources. We saw in Chapter 3 how to expand on process modeling by building a model of a STEM mixer. In that modeling, there were a number of activities (getting the name tag, wandering around, visiting with recruiters) that were modeled with basic modeling constructs such as SEIZE, DELAY, RELEASE. In addition, we saw how to use the DECIDE and LABEL modules to direct the flow of entities within a model. Chapter 3 also introduced the common statistical quantities produced by simulation models and provided an overview for how to plan for the execution of the model by determining the size of the sample for the experiments. In this chapter, the basic modeling concepts of Chapter 2 and 3 will be expanded on to allow you to model situations involving the movement of entities within a system. Specifically, we will introduce, through an expanded version of the STEM mixer, the notion of modeling locations in the physical world where processing occurs. This will be based on the use of the STATION module. Then, we will be able to model movement between locations (stations) via the transfer of entities by the ROUTE module. These modules will essentially replace the LABEL and GOTO LABEL modules of Chapter 3. Some additional animation elements will also be introduced. Finally, in additional models, we will see how to model processes that occur in parallel and how to share resources between activities. We will start with the enhanced STEM Career Mixer example.

4.1. Enhancing the STEM Career Mixer Example

In Chapter 3, we discussed the implementation of an Arena model for the STEM Career Fair Mixer of Example 3.4. In this section, we are going to embellish the system to add a few modeling issues that will make the model a bit more realistic. This will also allow for the introduction of the following new Arena modules.

- EXPRESSION
- ROUTE
- STATION
- STORAGE
- STORE
- UNSTORE

We will also re-examine how the CREATE module works and see how to collect additional performance metrics. Finally, we will learn how to control the underlying random number streams used when generating random variables within Arena.

The following issues will be addressed in the new model.

- In reality, those students that wander around before deciding to visit the recruiters actually visit the conversation area associated with the mixer. The organizer of the event has asked alumni to volunteer to be available within the conversation area to chat with students and discuss their career development. After all, it is a mixer. The organizer of the mixer would like to plan for the size of the conversation area.
- The STEM Fair is scheduled for 6 hours; however, currently, the simulation simply ends “abruptly” at 6 hours. How should we handle those students that are attending the mixer when the mixer ends? In this section, we will model a more realistic approach to ending the mixer and handling the students in progress.

After discussing with the STEM Fair organizer, the following additional facts were discovered. Generally, near the end of the scheduled mixer time, an announcement is made to everyone in the facility through the public address (PA) system that the mixer will be ending soon. For the purposes of this modeling enhancement, we can assume that the announcement goes out 15 minutes before the end of the mixer. At that time, we can assume the following:

- The doors to the mixture are closed to new arrivals. In other words, a closed sign goes up so that no new students are admitted to the mixer.
- Any students that were chatting within the conversation area finish up their conversations and then head for the exit, without visiting the recruiting stations.

4.1. Enhancing the STEM Career Mixer Example

- Any student walking to the name tag area, the conversation area or to the recruiting area proceed to the exit.
- Any students already at the recruiting stations, either in line waiting, or talking with the recruiter are allowed to finish up their visit and then depart the mixer.

Based on additional observations and discussion with the STEM mixer organizer, the following changes can be assumed:

- Because the distances between the stations in the hall are now important an enhanced drawing of the system has been made that better illustrates the standard layout that has been used in the past for the mixer. The distances shown in the drawing are rough estimates.
- New data suggest that 60% of the arriving students do not visit the conversation area, but instead go directly to the recruiting area to pick one of the two recruiting stations to visit, with equal probability.
- The remaining 40% of students will first visit the conversation area. The time spent within the conversation area is a little less than previously noted to exclude the time spent walking. The conversation time is triangularly distribution with a minimum of 10 minutes, a most likely value of 15 minutes, and a maximum value of 30 minutes. After having their conversations, 90% of the students decide to visit one of the two recruiting stations. The other 10% are too tired or timid and decide to leave.
- The speed of people walking can vary greatly, but prior data suggests that for short distances within and around buildings, people walk between 1 mile per hour and 3 miles per hour, with a most likely time of 2 miles per hour, triangularly distributed.

After further conversations with students who have attended past mixers, it was discovered that there really isn't a preference for visiting one of the recruiters first. Instead, after arriving to the recruiting area, students decide to visit the recruiter that has the least number of people (waiting and interacting with the recruiters). If both recruiter locations have the same total number of students, then the students show no preference between the two recruiters. After visiting the first recruiter, if they have already spent more than 45 minutes at the STEM fair, they decide to depart; otherwise, they visit the recruiter that they have not yet visited. After visiting both, they depart.

Because of these changes to the modeling assumptions, the STEM organizer would like to better understand the following aspects of potential system performance.

- When the PA announcement occurs, how many students on average are:
 - In the conversation area
 - Attending the MalWart recruiting station
 - Attending the JHBunt recruiting station

4. Modeling Systems with Processes and Basic Entity Flow

- How long after the 6-hour scheduled time does the mixer actually end? That is, since all students are permitted to finish up what they are currently doing when the PA announcement occurs, how long does it take for the hall to be completely empty?
- How many students, on average, are in the conversation area?
- The probability that a student departs because they have already spent more than 45 minutes at the mixer after visiting their first recruiting station.
- For students that do not visit the conversation area, what is the probability that their system time is less than 30 minutes?

Given this new description there are two major modeling issues that need to be addressed:

- 1) how to close the mixer and
- 2) modeling and visualizing the walking.

4.1.1. Turning Off a CREATE Module

In order to better model the closing of the mixer, we need to understand how to turn off a CREATE module. As shown in Figure 4.1, the CREATE module has a field labeled “Max Arrivals”. For every CREATE module there is an internal variable that counts how many creation events have occurred. During the simulation, this variable is automatically compared to the current value of the field labeled “Max Arrivals” and if the internal variable becomes greater than or equal to the current value of the field “Max Arrivals” then the CREATE module stops its creation process. For example, suppose that the value of Max Arrivals was 10. Then, only 10 creation events will occur.

By using a variable, say vMaxNumArrivals, the user can specify how many arrivals there should be. When you want to turn off the CREATE module just set the value of vMaxNumArrivals to 0. So, to close the door for the STEM mixer we will use another CREATE module that will cause the value of vMaxNumArrivals to be set to zero, at time 345 minutes (i.e. 15 minutes before the end of the mixer). This will also facilitate the collection of the additional performance measures.

4.1.2. Modeling Walking Time

To model the walking within the mixer, we need to translate the distance travelled into time. Since we are told the typical velocity for walking within a building for a person, we can randomly generate the velocity associated with a walking trip. Then, if we know the distance to walk, we can determine the time via the following relationship, where v is the speed (velocity), d is the distance, and t is the time.

$$v = \frac{d}{t}$$

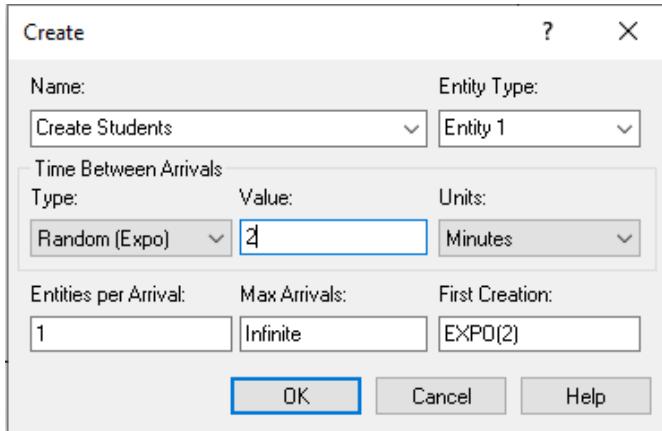


Figure 4.1.: Example CREATE module

Thus, we can randomly generate the value of v , and use the relationship to determine the time taken.

$$t = \frac{d}{v}$$

We know that the speed of people walking is triangularly distributed with a minimum of 1 mile per hour, a mode of 2 miles per hour, and a maximum of 3 miles per hour. This translates into a minimum of 88 feet per minute, a mode of 176 feet per minute, and a maximum of 264 feet per minute. Thus, we represent the speed as a random variable, Speed $\sim \text{TRIA}(88, 176, 264)$ feet per minute and if we know the distance to travel, we can compute the time.

We have the following distances between the major locations of the system:

- Entrance to Name Tags, 20 feet
- Name Tags to Conversation Area, 30 feet
- Name Tags to Recruiting Area (either JHBunt or MalMart), 80 feet
- Name Tags to Exit, 140 feet
- Conversation Area to Recruiting Area (either JHBunt or MalMart), 50 feet
- Conversation Area to Exit, 110 feet
- Recruiting Area (either JHBunt or MalMart) to Exit, 60 feet

For simplicity, we assume that the distance between the JHBunt and MalWart recruiting locations within the recruiting area can be ignored.

4. Modeling Systems with Processes and Basic Entity Flow

4.1.3. Using Expressions within an Arena Model

Within Arena mathematical expressions may be formed using combinations of integers, constants, attributes, variables, or random distributions. Conditional statements may also be used. Conditional expressions are evaluated as numerical expressions; a true condition is assigned a value of 1, and a false condition is assigned a value of 0. We have already used many expressions within the example models. Suppose we have some distribution (e.g. walking speed distribution) that we need to use in many places within the model. That is, we will need to enter the mathematical expression in many different fields within different modules within the model. Now suppose, after about a month of using the model, we decide that the expression needs to be updated because of some new data. Since the mathematical expression is used throughout the model, we need to find it everywhere that it is used. Of course, we could use the search and find capability associated with the Arena environment to make the edits; however, a much better approach is to use the EXPRESSION module found on the Advanced Process panel.

The EXPRESSION module permits the definition of a named expression. Then, the name of the expression can be used throughout the model, similar to how variables and attributes are used. The EXPRESSION module defines expressions and their associated values. An expression can contain any Arena-supported expression logic, including real values, Arena distributions, Arena or user-defined system variables, and combinations of these. Expressions can be single elements or one- or two-dimensional arrays. By defining and naming an expression, you can use that name throughout your model. Then, if there is an update of the expression, you need only edit the expression within the EXPRESSION module to change it everywhere that it is used in the model. Expressions are extremely useful, especially arrayed expressions. We will use an expression defined within the EXPRESSION module to represent the speed distribution within the new implementation of the model. As a naming convention, I append the letter “e” to the beginning of any of my expressions. For example, we will use the name “eWalkingSpeedRV” to represent the walking speed within the new implementation. By appending the letter “e” to the name of your expressions, you will know that it is a defined expression when you see it used within various modules.

4.1.4. Introducing the STATION, ROUTE, STORAGE, and SET Modules

Because the model stakeholders desire improved animation, we will also need to introduce some new Arena constructs: STATION, ROUTE, and STORAGE.

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.

4.1. Enhancing the STEM Career Mixer Example

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.

The STATION module is found on the Advanced Transfer panel. See Figure 4.2 for the edit by dialog view of the STATION module. For our current purposes, we can think of a STATION module as a fancy label. A STATION denotes a location within the model to and from which entity instances can be transferred. Stations work very much like labels; however, special entity transfer modules are needed to send the entities to and from stations. Those modules are found on the Advanced Transfer panel, one of which is the ROUTE module, as shown in Figure 4.3. For our current purposes, a ROUTE module can be thought of as a Goto Label module, where we specify a station as the destination rather than a named label. ROUTE modules allow for the specification of a time delay associated with the transfer from the current location to the destination. The nice thing about this is that there are convenient animation constructs available for showing the entities as they experience the transfer delay. We will use this capability to show the people walking between the areas within the STEM mixer. The other modules associated with the Advanced Transfer panel will be discussed in Chapter 7.

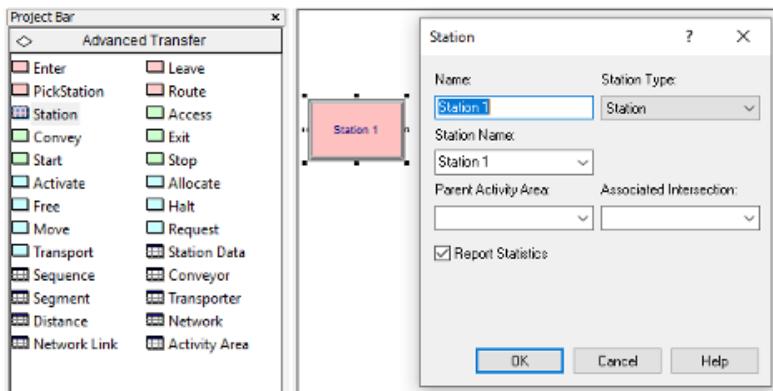


Figure 4.2.: The STATION module

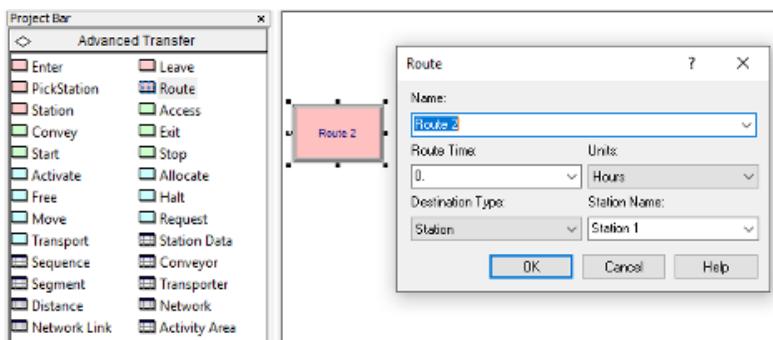


Figure 4.3.: The ROUTE module

Finally, we will use the STORAGE module found on the Advanced Process panel to model the

4. Modeling Systems with Processes and Basic Entity Flow

conversation areas associated with the STEM mixer. Figure 4.4 shows the STORAGE module as a data definition module on the Advanced Process panel. The STORAGE module is also a named location within the model. The STORAGE module simply creates and defines a storage that can be used within the model. The primary purpose of a storage is for animation; however, every storage has a special variable accessed through the function, NSTO(Storage Name), that returns the current number of entities associated with the storage. To place an entity in a storage, we use the STORE module and to take an entity out of storage, we use the UNSTORE module. The STORE and UNSTORE modules are also found on the Advanced Process Panel as illustrated in Figures 4.5 and 4.6. The STORE module automatically increments the number of entities in the associated storage by one and the UNSTORE module automatically decrements the number of entities in the associated storage by one. Thus, the variable, NSTO(Storage Name) can be used to collect statistics on the number of entities associated with the storage at any time. The nice thing about storages is that animated storages can be associated with the storage in order to display the entities currently in the storage during the animation. This capability could be used to show the students that are in the conversation area. In fact, a storage can be useful in animating the time that an entity is in the time delayed state.

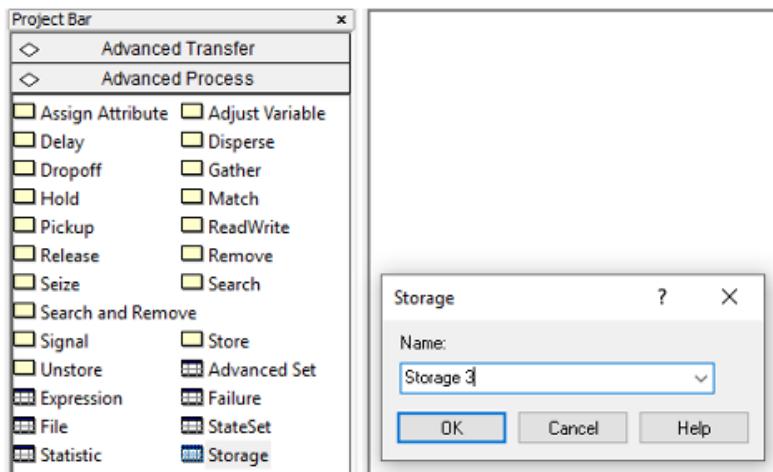


Figure 4.4.: The STORAGE module

The last new modeling construct that we will use within the implementation of the enhanced STEM Career Fair Mixer model is that of sets. Sets represent a set of similar elements held in an indexed list. Sets are an incredibly useful construct for implementing many useful modeling situations, especially the random selection of elements from the list. We will demonstrate the use of set by introducing tally sets (sets that hold tally statistics) and a station set (a set that holds a list of station). In the case of the station set, we will randomly select a station from the set and then send an entity to the station.

4.1. Enhancing the STEM Career Mixer Example

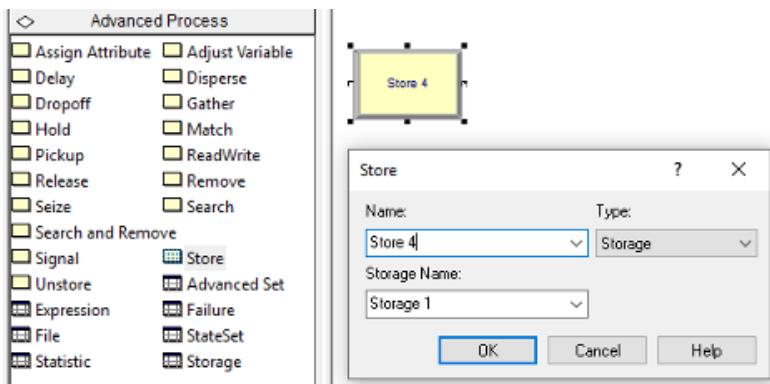


Figure 4.5.: The STORE module

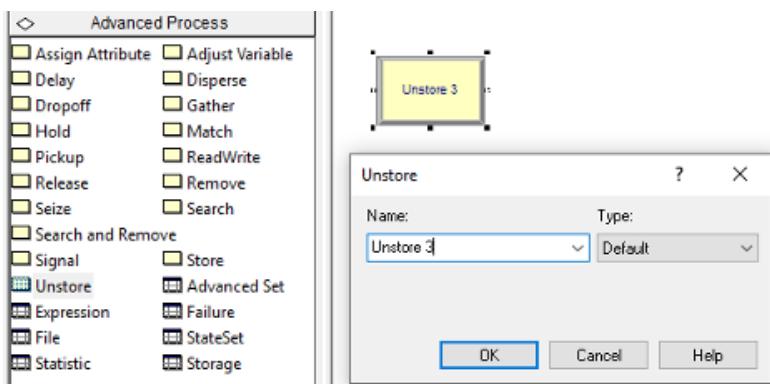


Figure 4.6.: The UNSTORE module

4.1.5. Controlling Randomness by Specifying Stream Numbers

Random number generators in computer simulation languages come with a default set of streams that divide a sequence of random numbers into independent sets (or sequences). 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 using simulation languages such as Arena 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 pseudo random numbers today as it did yesterday and the day before, etc. The answer to

4. Modeling Systems with Processes and Basic Entity Flow

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.

Since a random number stream is a sub-sequence of pseudo-random numbers that start at particular place with a larger sequence of pseudo-random numbers, it is useful to have a starting point for the sequence. The starting point of a sequence of pseudo-random numbers is called the *seed*. A seed allows us to pick a particular stream. Having multiple streams is useful to assign different streams to different sources of randomness within a model. This facilitates the control of the use of pseudo-random numbers when performing experiments. Every distribution in Arena has an extra *optional* parameter for the stream. For example, EXPO(mean, stream#), LOGN(mean, sd, stream#), DISC(cp1, v1, cp2, v2, ..., 1.0, vn, stream#), where stream# is some integer representing the stream you want to use for that distribution. Arena associates an integer number, 1, 2, 3, etc. with the seed associated with a particular random number stream. For all intents and purposes, we can assume that each stream produces independent random numbers.



A random number stream is a sub-sequence of pseudo-random numbers that start at particular place with a larger sequence of pseudo-random numbers. The starting point of a sequence of pseudo-random numbers is called the *seed*. A seed allows us to pick a particular stream. Having multiple streams is useful to assign different streams to different sources of randomness within a model. Streams can be further divided into sub-streams. This facilitates the control of the use of pseudo-random numbers when performing experiments.

To specify the stream in the CREATE module, use the "Expression" option for the type and then specify the distribution, e.g. EXPO(mean, 2). Use the expression option to specify the exact required expression. To control the stream associated with a DECIDE, use the DISC() function to assign an attribute, and then use a DECIDE module to check that attribute. For example:

```
// 40% chance for path 1, 60% chance for path 2, using stream 9
ASSIGN: myFlag = DISC(0.4, 1, 1.0, 2, 9)
DECIDE: If myFlag == 1,
    go to path 1
Else
    goto path 2
```

To prepare the STEM Mixer model for better control of the underlying randomness due to random number generation, the following stream assignments should be made.

- Arrival process, stream 1

- Name tag activity, stream 2
- Choice to visit conversation area, stream 3
- Choice between conversation areas, stream 4
- Conversation time, stream 5
- Deciding to visit recruiters after visiting conversation area, stream 6
- Choice between MalMart and JHBunt, stream 7
- MalMart interaction time, stream 8
- JHBunt interaction time, stream 9
- Walking speed, stream 10

Using streams to control the randomness will allow us to better implement a variance reduction technique called common random numbers when we make comparisons between system configurations. This will be discussed in Section 4.4.



If the simulation uses the same underlying pseudo-random numbers, **how come my simulation results are sometimes unexpectedly different?**. Sometimes by changing the order of modules you change the sequence of random numbers that are assigned to various things that happen in the model (e.g. entity attribute, service times, paths taken, etc.). Now, the result can sometimes be radically different if different random numbers are used for different purposes. By using streams dedicated to different random processes, you reduce this possibility and increase the likelihood that two models that have different configurations will have differences due to the change and not due to the random numbers used.

Further details about how to generate random numbers and random variates are provided in Appendix A. The next section presents the pseudo-code for the revised STEM Mixer example.

4.1.6. Pseudo-code for the Revised STEM Mixer Example

Now we are ready to update the pseudo-code from Chapter 3 to address the previously discussed enhancements. The first thing to do is to define in the pseudo-code the modeling constructs that we will utilize. For this situation, we will define six new variables. The variable vClosed will be used to indicate that the mixer is closed. Then, we will have two new variables to keep track of how many students are visiting the MalWart and JHBunt recruiters, vNumAtMalWart and vNumAtJHBunt. Finally, we have three variables relate to the closing of the mixer: vMaxNumArrivals (to shut off the CREATE module), vWarningTime (to represent the time interval to the end of the mixer) and vMixerTimeLength (to represent the total time available for the mixer). These variables are defined in the following pseudo-code.

4. Modeling Systems with Processes and Basic Entity Flow

```
// Model Definition Pseudo-Code

ATTRIBUTES:
myType // represents the type of the student (1= recruiters first, 2 = conversation first, 3 = timid).
myArrivalTime // the time that the student arrived to the mixer
myFirstRecruiter // the recruiter visited first
myNumVisits // the number of recruiters visited

VARIABLES:
vNumInSys // represents the number of students attending the mixer at any time t
vClosed = 0// represents that the STEM fair has closed its doors, 1 = closed, 0 = open
vNumAtJHBunt // number of students visiting JHBunt recruiters
vNumAtMalWart // number of students visiting MalWart recruiters
vMaxNumArrivals = 10000// used to control creation of students
vWarningTime = 15 // minutes, used to control when to close the STEM Mixer
vMixerTimeLength = 360 minutes

EXPRESSIONS:
eDoorClosingTime = vMixerTimeLength - vWarningTime
eWalkingSpeedRV = TRIA(88, 176, 264, 10)
eEntranceToNameTagsTimeRV = 20.0/eWalkingSpeedRV
eNameTagsToConverseAreaTimeRV = 30.0/eWalkingSpeedRV
eNameTagsToRecruitingAreaTimeRV = 80.0/eWalkingSpeedRV
eNameTagsToExitTimeRV = 140.0/eWalkingSpeedRV
eConverseAreaToRecruitingAreaTimeRV = 50.0/eWalkingSpeedRV
eConverseAreaToExitTimeRV = 110.0/eWalkingSpeedRV
eRecruitingAreaToExitTimeRV = 60.0/eWalkingSpeedRV

RESOURCES:
rMalWartRecruiters, 2 // the MalWart recruiters with capacity 2
rJHBuntRecruiters, 3 // the JHBunt recruiters with capacity 3

STORAGES:
stoConversationArea //use to show students in the conversation area

STATIONS:
staEntrance // the entrance to the mixer
staNameTag // the location to get name tags
staConverseArea // the area for holding conversations
staRecruitingArea // the area for visiting recruiters
staJHBunt // the JHBunt recruiting location
staMalWart // the MalMart recruiting location
staExit // the location to exit the mixer
```

4.1. Enhancing the STEM Career Mixer Example

SETS:

```
RecruiterStationSet (staJHBunt, staMarWart) // hold stations for picking between stations
SystemTimeSet(Type1SysTime, Type2SysTime, Type3SysTime) // to more easily collect statistics
```

In addition to variables, we have defined nine expressions, eight of which are related to the walking and time taken for movement between locations. A STORAGE has been defined to track the number of students in the conversation area and seven stations have been defined to represent locations. Finally, two sets are defined. The RecruiterStationSet holds the stations representing the JHBunt recruiting location and the MalWart recruiting location. The SystemTimeSet is a tally set that holds the tally statistical variables for collecting statistics by type of student. Now, we are ready to represent the model logic.

```
// Creating the Students
CREATE vMaxNumArrivals of students every EXPO(2,1) minutes with the
first arriving at time EXPO(2,1)
STATION staEntrance
BEGIN ASSIGN
    myArrivalTime = TNOW
    vNumInSys = vNumInSys + 1
END ASSIGN
ROUTE to staNameTag, time = eEntranceToNameTagsTimeRV
```

The logic for creating the students for the mixer is very similar to what was done in Chapter 3. The changes include having a variable, vMaxNumArrivals, to use to turn off the CREATE module and using a STATION to represent the entrance to the mixer. At the station, we assign the arrival time of the student for statistical collection and increment the number of students attending the mixer. Then, we use a ROUTE module to send the student to the name tag area using the time expression for modeling the walking from the entrance to the name tag area.

```
// Getting a Name Tag
STATION staNameTag
DECIDE IF vClosed == 1
    ROUTE to staExit, time = eNameTagsToExitTimeRV
END DECIDE
DELAY UNIF(15,45, 2) seconds to get name tag
ASSIGN myType = DISC(0.60, 1, 1.0, 2, 3) // 1 = recruiting only, 2 = conversation area
DECIDE If myType == 1 // recruiting station only student
    ROUTE to staRecruitingArea, time = eNameTagsToRecruitingAreaTimeRV
ELSE // myType = 2, conversation first student
    ROUTE to staConverseArea, time = eNameTagsToConverseAreaTimeRV
END DECIDE
```

4. Modeling Systems with Processes and Basic Entity Flow

The logic for getting a name tag has been updated to include sending students that were headed to the name tag area to the exit if the mixer closing warning occurred during their walk to the name tag area. Notice the specification of the stream number in the UNIF() distribution representing the time to get a name tag. Then, a DISC() distribution is used to determine the type of student and then the student is sent to the appropriate area via ROUTE modules. The pseudo-code for the conversation area, has the same check for closing during the walk to the location. Then, we see the use of the STORE and UNSTORE constructs to track how many students are in the conversation area during the delay for the conversation. Sending the students to the exit or to recruiting is used by assigning their type and then using a DECIDE to send them to the appropriate station.

```
// Visiting the Conversation Area
STATION staConverseArea
DECIDE IF vClosed == 1
    ROUTE to staExit, time = eConverseAreaToExitTimeRV
END DECIDE
STORE stoConversationArea
DELAY TRIA(10, 15, 30, 5) minutes for conversation
UNSTORE stoConversationArea
ASSIGN myType = DISC(0.9, 2, 1.0, 3, 6)
DECIDE If myType == 2 // recruiting station student
    ROUTE to staRecruitingArea, time = eConverseAreaToRecruitingAreaTimeRV
ELSE // myType = 3, timid student
    ROUTE to staExit, time = eConverseAreaToExitTimeRV
END DECIDE
```

After arriving to the recruiting area, we check if the mixer has closed and if so, send the student to the exit. Then, the logic checks the variables used to keep track of the number of students visiting JHBunt and MalWart are checked to determine which recruiter to visit. Notice the use of the DISC() distribution function in the case of ties. An attribute, myFirstRecruiter, is used to hold the choice, and then the appropriate station is selected from the recruiter station set based on the attribute. The use of a randomly assigned attribute to index into a set is a very useful pattern. Besides indexing into a set, this same idea can be used to randomly pick a value from an array or an expression.

```
// Visiting the Recruiting Area
STATION: staRecruitingArea
DECIDE IF vClosed == 1
    ROUTE to staExit, time = eRecruitingAreaToExitTimeRV
END DECIDE
DECIDE IF vNumAtJHBunt == vNumAtMalWart
    ASSIGN myFirstRecruiter = DISC(0.5, 1, 1.0, 2, 7)
ELSE IF vNumAtJHBunt < vNumAtMalWart
```

4.1. Enhancing the STEM Career Mixer Example

```

ASSIGN myFirstRecruiter = 1
ELSE
    ASSIGN myFirstRecruiter = 2
ENDIF
ROUTE with 0 delay, to RecruiterStationSet(myFirstRecruiter)

```

The pseudo-code for attending the MalWart or JHBunt recruiting stations is essentially the same. First we increment the number of students that are visiting and then process the students at the station.

```

// Attending MalWart
STATION: staMalWart
ASSIGN vNumAtMalWart = vNumAtMalWart + 1
SEIZE 1 MalWart recruiter
DELAY for EXPO(3,8) minutes for interaction
RELEASE 1 MalWart recruiter
ASSIGN vNumAtMalWart = vNumAtMalWart - 1
ASSIGN myNumVisits = myNumVisits + 1
DECIDE IF (myNumVisits == 1) AND (TNOW - myArrivalTime) > 45
    ASSIGN my45MinuteFlag = 1
    ROUTE to staExit, time = eRecruitingAreaToExitTimeRV
ELSE IF myNumVisits == 2
    ROUTE to staExit, time = eRecruitingAreaToExitTimeRV
ELSE
    ROUTE with 0 delay to staJHBunt
END DECIDE

```

The logic for determining which recruiter is visited next is interesting. Using an attribute, myNumVisits, allows us to determine if one or both recruiters have been visited. In the case of only one visit, we can determine whether or not the student has been at the mixer more than 45 minutes. If so, an attribute is used as an indicator variable to denote this event for later statistical processing. If both stations have been visited, then the student exits the mixer. If only one recruiter has been visited, then the student is sent to the opposite recruiter.

```

// Attending JHBunt
STATION: staJHBunt
ASSIGN vNumAtJHBunt = vNumAtJHBunt + 1
SEIZE 1 JHBunt recruiter
DELAY for EXPO(6,9) minutes for interaction
RELEASE 1 JHBunt recruiter
ASSIGN vNumAtJHBunt = vNumAtJHBunt - 1
ASSIGN myNumVisits = myNumVisits + 1

```

4. Modeling Systems with Processes and Basic Entity Flow

```

DECIDE IF (myNumVisits == 1) AND (TNOW -- myArrivalTime) > 45
    ASSIGN my45MinuteFlag = 1
    ROUTE to staExit, time = eRecruitingAreaToExitTimeRV
ELSE IF myNumVisits == 2
    ROUTE to staExit, time = eRecruitingAreaToExitTimeRV
ELSE
    ROUTE with 0 delay to staMalWart
END DECIDE

```

The exit area is primarily used to collect statistics on departing students. There are three updates to note. First, we can collect statistics on the probability that the students that do not visit the conversation have a system time that is less than or equal to 30 minutes. We use an expression (mySysTime <= 30) to collect on a logical indicator of this event. Secondly, we used a tally set to record the system times by student type (myType). Lastly, we director record on the indicator attribute, my45MinuteFlag, to collect the probability that the student exited after visiting only one recruiter because they had already been at the mixer for more than 45 minutes.

```

// Exiting and Collecting Statistics
STATION: staExit
ASSIGN: vNumInSys = vNumInSys - 1
ASSIGN: mySysTime = TNOW -- myArrivalTime
DECIDE IF myType == 1 // no conversation students
    RECORD (mySysTime <= 30)
END DECIDE
RECORD mySysTime, as System Time regardless of type
RECORD mySystTime, using SystemTimeSet by myType
RECORD my45MinuteFlag
DISPOSE

```

Now we have to model the closing of the STEM Mixer. As previously mentioned, we can do this by creating a logical entity that is used to turn off the CREATE module governing the creation of the students. We create one entity at the door closing time and close the door by changing the global variable, vClosed. In addition, we collect observation based statistics via RECORD statement that records the values of the variables *at this particular instance in time*. These are observation based statistics, *not* time persistent, because we are observing the random value at a particular instance of time (not over time).

```

// Model Pseudo-Code Listing for Closing the Mixer
CREATE 1 entity, at time eDoorClosingTime
ASSIGN vMaxNumArrivals = 0
ASSIGN vClosed = 1
RECORD Expressions // observe variables at door closing time

```

4.1. Enhancing the STEM Career Mixer Example

```

NSTO(stoConversationArea)
vNumAtMalWart
vNumAtJHBunt
vNumInSys
END RECORD
DISPOSE

```

Now that all the pseudo-code is specified, implementing the model in Arena is very straightforward. We implement these ideas in the next section.

4.1.7. Implementing the Revised STEM Mixer Model in Arena

To implement this model within Arena, we can follow the pseudo-code starting with the model definitions. Figure 4.7 and 4.8 show the revised attributes and variables for the Arena model. It is best practice to define your modeling constructs first before using them in the Arena flow chart area. This allows you to select the construct from drop-down boxes or from within the expression builder rather than directly typing in the construct. This reduces the number of typos and other syntax errors.

Attribute - Basic Process						
	Name	Comment	Rows	Columns	Data Type	Initial Values
1	myArrivalTime	holds when students arrive at the mixer			Real	0 rows
2	myType	1= recruiters first, 2 = conversation first, 3 = timid			Real	0 rows
3	mySysTime	total time spent at the mixer			Real	0 rows
4	myNumVisits	the number of recruiters visited			Real	0 rows
5	myFirstRecruiter	the recruiter visited first			Real	0 rows
6 ►	my45MinuteFlag	indicates that visited 1 recruiter and left because of 45 minutes in mixer			Real	0 rows

Double-click here to add a new row.

Figure 4.7.: Defining the attributes for the revised STEM Mixer example

Variable - Basic Process									
	Name	Comment	Rows	Columns	Data Type	Clear Option	File Name	Initial Values	Report Statistics
1	vNumInSys	tracks number of students in the system			Real	System		0 rows	<input checked="" type="checkbox"/>
2	vClosed	indicates if mixer is closing			Real	System		0 rows	<input type="checkbox"/>
3	vNumAtJHBunt	tracks number of students visiting JHBunt recruiters			Real	System		0 rows	<input type="checkbox"/>
4	vNumAtMalMart	tracks number of students visiting MalMart recruiters			Real	System		0 rows	<input type="checkbox"/>
5 ►	vWaningTime	in minutes			Real	System		1 rows	<input type="checkbox"/>
6	vMixerTimeLength	in minutes			Real	System		1 rows	<input type="checkbox"/>
7	vMaxArrivals	used to shut off student arrivals			Real	System		1 rows	<input type="checkbox"/>

Double-click here to add a new row.

Figure 4.8.: Defining the variables for the revised STEM Mixer example

Notice how the comments section of the spreadsheet view of the ATTRIBUTE and VARIABLE modules provides an easy way to specify to the modeler the purpose and use of the attribute or variable. In Figure 4.9, we see the definitions of the expressions used throughout the model.

4. Modeling Systems with Processes and Basic Entity Flow

The EXPRESSION module also allows for a comment to indicate information about the expression. In the figure, we see the eEntranceToNameTagsTimeRV expression that implements the distance (20 feet) divided by the walking speed expression. All of the routing time distributions were defined in terms of eWalkingSpeedRV. Thus, if the distribution changes, it need only be edit in a single expression. Figure 4.10 presents the station data and Figure 4.11 shows the definition of a set to hold the two recruiting stations. The set definition module available on the Advanced Process panel was used to define the recruiting station set.

Expression - Advanced Process						
	Name	Comment	Rows	Columns	Data Type	File Name
1	eWalkingSpeedRV	typical walking speed of a person			Native	1 rows
2 ►	eEntranceToNameTagsTimeRV	time to travel from entrance to name tags			Native	1 rows
3	eNameTagsToConverseAreaTimeRV	time to travel from name tags to conversation area			Native	1 rows
4	eNameTagsToRecruitingAreaTimeRV	time to travel from name tags to recruiting area		Expression Values		
5	eNameTagsToExitTimeRV	time to travel from name tags to exit				
6	eConverseAreaToRecruitingAreaTimeRV	time to travel from conversation area to recruiting area		20.0/eWalkingSpeedRV		
7	eConverseAreaToExitTimeRV	time to travel from conversation area to exit				
8	eRecruitingAreaToExitTimeRV	time to travel from recruiting area to exit			Native	1 rows
9	eDoorClosingTime	time that mixer is scheduled to close			Native	1 rows

Figure 4.9.: Defining the expressions for the revised STEM Mixer example

Station Data - Advanced Transfer					
	Symbol Number	Name	Associated Intersection	Parent Activity Area	Report Statistics
1 ►		staEntrance			<input checked="" type="checkbox"/>
2		staNameTags			<input checked="" type="checkbox"/>
3		staConversationArea			<input checked="" type="checkbox"/>
4		staRecruitingArea			<input checked="" type="checkbox"/>
5		staJHBunt			<input checked="" type="checkbox"/>
6		staMaWart			<input checked="" type="checkbox"/>
7		staExit			<input checked="" type="checkbox"/>

Double-click here to add a new row.

Figure 4.10.: Defining the stations for the revised STEM Mixer example

Advanced Set - Advanced Process				
	Name	Set Type	Member Definition Method	Members
1 ►	RecruiterStationSet	Other	Manual List	2 rows
Members				
	Other			
1	staJHBunt			
2	staMaWart			

Double-click here to add a new row.

Figure 4.11.: Defining the station set for the revised STEM Mixer example

To implement the flow chart modules, we can start with the creation of the students and the

4.1. Enhancing the STEM Career Mixer Example

logical entity to shut off the creation of students at the appropriate time.

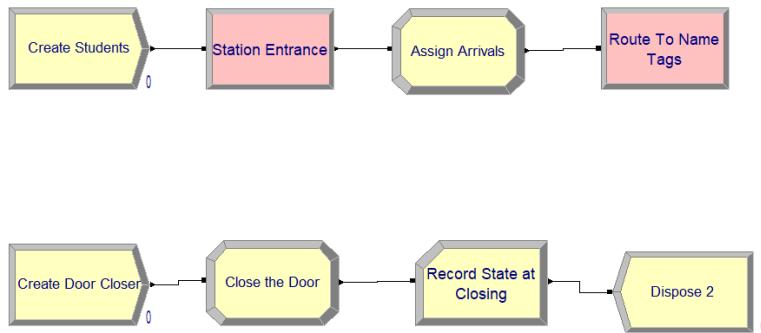


Figure 4.12.: Creating the students and closing the doors

Create - Basic Process										
	Name	Entity Type	Type	Value	Expression	Units	Entities per Arrival	Max Arrivals	First Creation	
1	Create Students	Entity 1	Expression	2	EXPO(2,1)	Minutes	1	vMaxArrivals	EXPO(2,1)	
2 ►	Create Door Closer	Entity 1	Random (Expo)	1	1	Minutes	1	1	eDoorClosingTime	

Figure 4.13.: Data for CREATE modules

Figure 4.12 shows the flow chart modules for creating the students and the door closing entity, which map almost directly to the pseudo-code listings. Figure 4.13 shows the data view of the CREATE modules. Notice the use of the stream number in specifying the mean time between arrivals of the students and the use of the variable, vMaxArrivals, in the “Max Arrivals” field. The expression, eDoorClosingTime, is used to specify the time of the first creation for the door closing entity. Why use an expression for this time? As noted in Figure 4.9, the eDoorClosingTime expression is computed from the two variables, vWarningTime and vMixerTimeLength. Defining these as parameters to the model facilitates their use when performing experiments. This is also good model building practice. To the extent possible define parameters as variables or expressions. Do not embed magic numbers (undocumented constants) within the text fields of the dialog boxes. After being created, the students go to the name tag area as illustrated in Figure 4.14. As noted in the pseudo-code, first we check if the mixer closed during the time that it took for the entity to get to the name tag area. If the mixer has not closed, the student decides whether or not to directly visit the recruiting area or the conversation area.

Those students that first go to the conversation area proceed to the logic illustrated in Figure 4.15. The other students go directly to the logic shown in Figure 4.16. Notice the use of the STORE and UNSTORE modules in Figure 4.16.

Figure 4.16 shows the use of the DECIDE module to implement the logic associated with determining which recruiter is visited first. If there is a tie between the two recruiters, one of stations associated with the recruiters is randomly selected; otherwise, the station with the lower number of students is selected. Figure 4.17 shows the data view for all of the ROUTE modules in the model. Notice that the already discussed expressions are used in each of the ROUTE modules. In addition, the ROUTE module defined on line 11 implements the random selection of

4. Modeling Systems with Processes and Basic Entity Flow

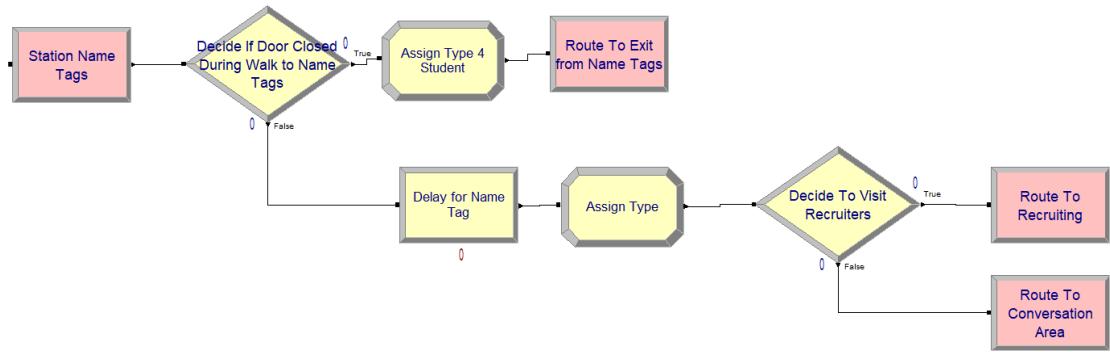


Figure 4.14.: Name tag station and modules

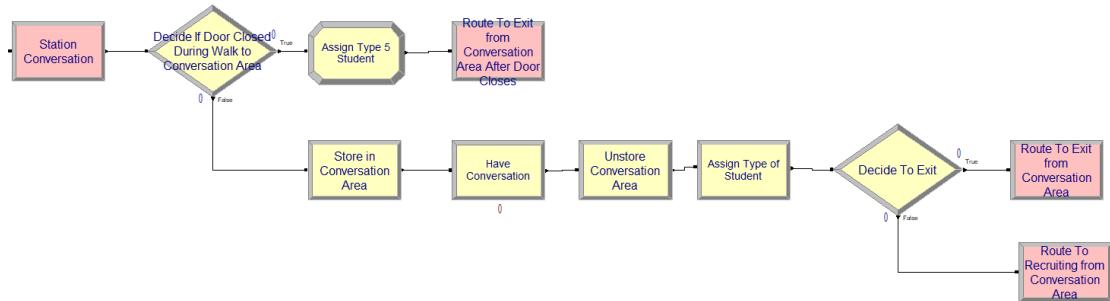


Figure 4.15.: Conversation area station and modules

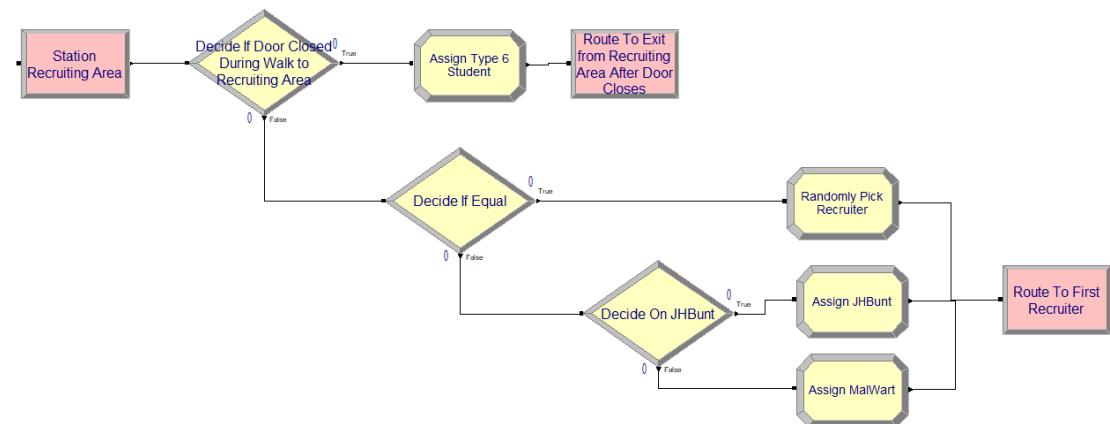


Figure 4.16.: Recruiting area station and modules

4.1. Enhancing the STEM Career Mixer Example

the station to visit by using the MEMBER() function. The Arena MEMBER(set name, set index) function will return the member of the set indicated by the index of the set member. In this case, the RecruiterStationSet is indexed into by the attribute myFirstRecruiter, which was assigned previously in one of the ASSIGN modules of Figure 4.16.

Route - Advanced Transfer						
	Name	Route Time	Units	Destination Type	Station Name	Expression
1	Route To Name Tags	eEntranceToNameTagsTimeRV	Minutes	Station	staNameTags	
2	Route To Recruiting	eNameTagsToRecruitingAreaTimeRV	Minutes	Station	staRecruitingArea	
3	Route To Conversation Area	eNameTagsToConverseAreaTimeRV	Minutes	Station	staConversationArea	
4	Route To Recruiting from Conversation Area	eConverseAreaToRecruitingAreaTimeRV	Minutes	Station	staRecruitingArea	
5	Route To Exit from Conversation Area	eConverseAreaToExitTimeRV	Minutes	Station	staExit	
6	Route To Exit From MalWart Recruiting Area	eRecruitingAreaToExitTimeRV	Minutes	Station	staExit	
7	Route To Exit from Name Tags	eNameTagsToExitTimeRV	Minutes	Station	staExit	
8	Route To Exit from Conversation Area After Door Closes	eConverseAreaToExitTimeRV	Minutes	Station	staExit	
9	Route To JHBunt	0.	Minutes	Station	staJHBunt	
10	Route To Exit from Recruiting Area After Door Closes	eRecruitingAreaToExitTimeRV	Minutes	Station	staExit	
11	Route To First Recruiter	0.	Minutes	Expression	Station 1	MEMBER(RecruiterStationSet,myFirstRecruiter)
12	Route To Exit From JHBunt Recruiting Area	eRecruitingAreaToExitTimeRV	Minutes	Station	staExit	
13 ►	Route To MalWart	0.	Minutes	Station	staMalWart	

Figure 4.17.: Data view for ROUTE modules

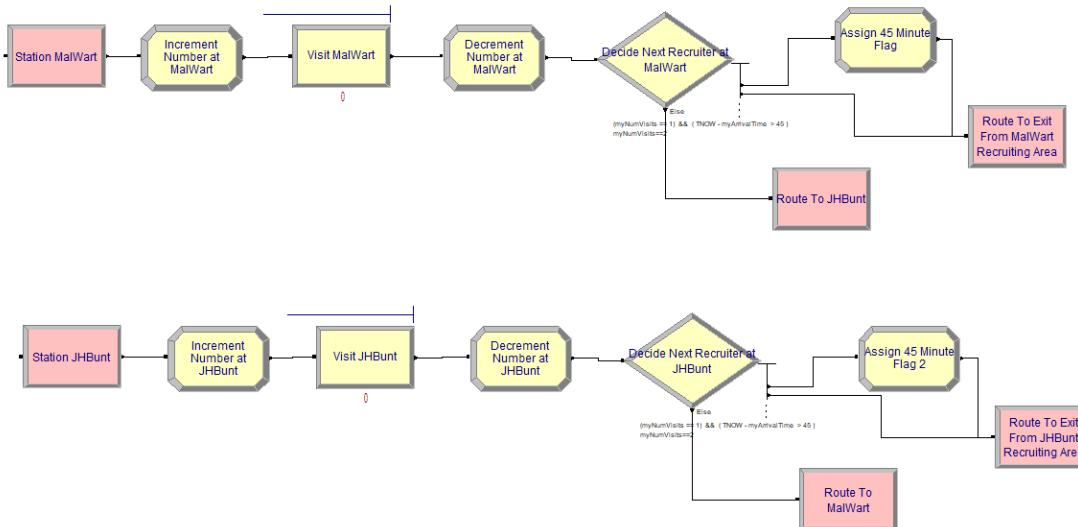


Figure 4.18.: JHBunt and MalWart recruiting station modules

After selecting the recruiter to visit first, the students will go to either of the two stations shown in Figure 4.18. The logic associated with both stations is essentially the same. First the number of students at the recruiter is incremented, the student enters the recruiting process, and then the number of students at the recruiter is decremented. The entries for the two PROCESS modules show in Figure 4.18 are shown in Figure 4.19. Finally, an N-way by condition is used to implement the DECIDE logic for exiting or visiting the other recruiter. Notice that the distributions associated with the delay have been updated to utilize the required stream numbers.

After visiting the recruiter area, the students will depart. Figure 4.20 shows the layout of the modules to collect statistics on departing students. This logic has been simplified, as compared

4. Modeling Systems with Processes and Basic Entity Flow

Process - Basic Process											
	Name	Type	Action	Priority	Resources	Delay Type	Units	Allocation	Expression	Report Statistics	
1	Delay for Name Tag	Standard	Delay	Medium(2)	0 rows	Expression	Seconds	Value Added	UNIF(15,45, 2)	<input checked="" type="checkbox"/>	
2	Have Conversation	Standard	Delay	Medium(2)	0 rows	Expression	Minutes	Value Added	TRIA(10,15,30,5)	<input checked="" type="checkbox"/>	
3	Visit MalWart	Standard	Seize Delay Release	Medium(2)	1 rows	Expression	Minutes	Value Added	EXPO(3,8)	<input checked="" type="checkbox"/>	
4 ►	Visit JHBunt	Standard	Seize Delay Release	Medium(2)	1 rows	Expression	Minutes	Value Added	EXPO(6,9)	<input checked="" type="checkbox"/>	

Figure 4.19.: STEM Mixer PROCESS modules

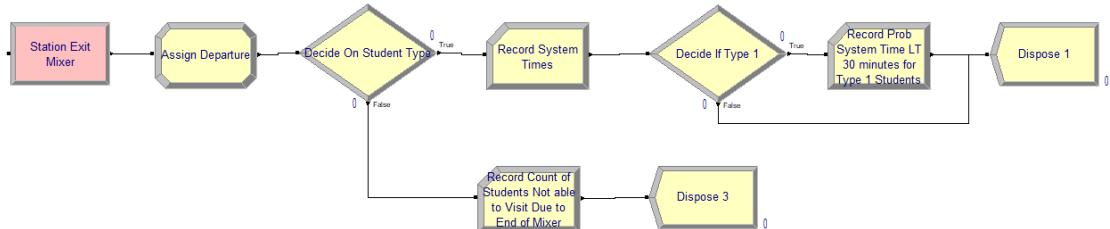


Figure 4.20.: STEM Mixer exit station modules

to, what was presented in Chapter 3 because of the use of tally sets. Figure x15 shows that a record can be defined that can collect statistics on the system time by type of student. This option is implemented by checking the “Record in Set” option for the RECORD module, designating the set containing the created tally variables, and defining the attribute that can be used to select the correct member from the set via the “Set Index” field.

Running the model for 30 replications, we achieve the results shown in Figure 4.22 and Figure 4.23.

Figure 4.23 shows the system times for the different types of students. A type 1 student goes directly to the recruiting area. A type 2 student first visits the conversation area before visiting the recruiting area. A type 3 student first visits the conversation area and then decides to directly depart the mixer. Figure 4.22 shows that there are on average about 15.7 students in the system when the closing starts. The probability that type 1 students spend less than 30 minutes attending the fair is about 77 percent on average.

In this section, we learned about the Arena elements: EXPRESSION, ROUTE, STATION, STORAGE, STORE, and UNSTORE. In particular, we saw how we can define expressions and use them throughout the model and represent the simple transfer of entities between locations with a time delay. In the next section, we build on some prior concepts, especially attributes, variables, and expressions to see how these concepts can be put to use within Arena. So far, our examples have shown situations where entities flow essentially forward through the model, perhaps taking different paths. In the next section, we illustrate how entities can be used to loop through Arena modules to implement coding logic. In addition, we see how arrays can facilitate model building and additional methods for organizing the modules within the flow chart area.

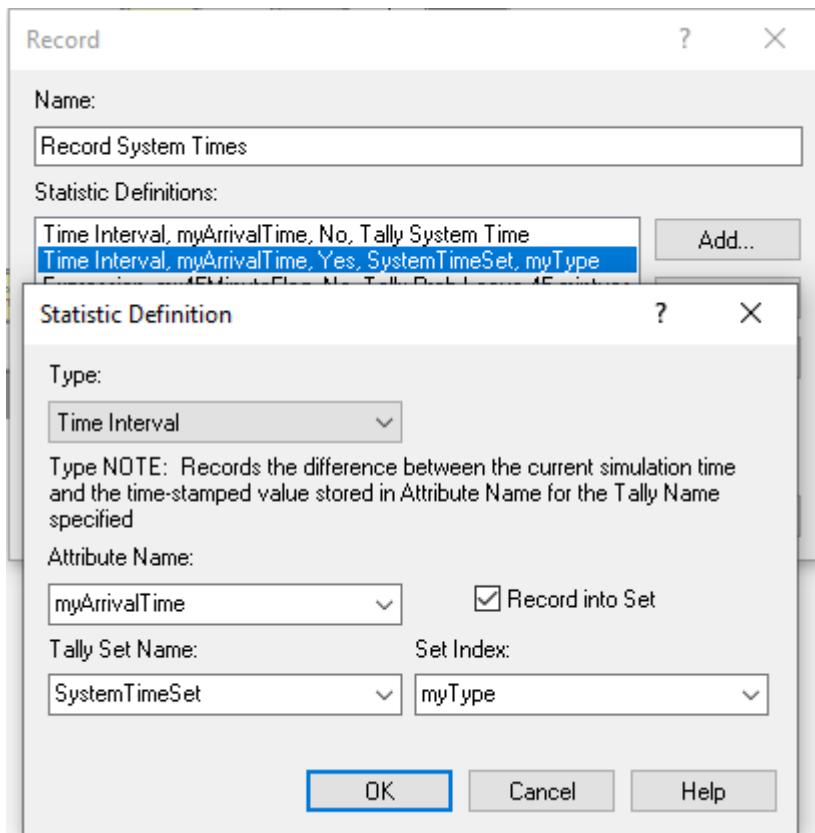


Figure 4.21.: Recording the system times with a tally set

4. Modeling Systems with Processes and Basic Entity Flow

Expression	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Num at JHBunt at Closing	9.1667	2.46	0.00	25.0000	0.00	25.0000
Num at MailMart at Closing	2.5000	0.85	0.00	8.0000	0.00	8.0000
Num Conversing at Closing	3.3333	0.76	0.00	8.0000	0.00	8.0000
Num in Sys at Closing	15.7000	2.25	6.0000	31.0000	6.0000	31.0000
Tally Prob Leave 45 mintues	0.00322337	0.00	0.00	0.02732240	0.00	1.0000
Type 1 Prob LT 30 min	0.7704	0.07	0.3143	0.9890	0.00	1.0000
Interval	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Tally System Time	27.6008	2.43	19.7176	44.7036	1.4594	101.64
Counter						
Count	Average	Half Width	Minimum Average	Maximum Average		
Count Type 1	103.67	4.00	80.0000	129.00		
Count Type 2	58.7333	2.70	44.0000	71.0000		
Count Type 3	6.9333	0.86	2.0000	12.0000		
Count Type 4 Door Closed Walking to Tags	1.0333	0.07	1.0000	2.0000		
Count Type 5 Door Closed Walking to Conversation	0.1667	0.17	0.00	2.0000		
Count Type 6 Door Closed Walking to Recruiting	3.5000	0.70	0.00	8.0000		
Num of Student Visits	169.33	5.01	132.00	189.00		

Figure 4.22.: Results for closing statistics

None	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Type1SysTime	21.1244	2.45	12.9001	37.6117	1.4594	84.4351
Type2SysTime	40.0047	2.66	31.4039	58.7734	12.5282	101.64
Type3SysTime	19.1402	0.53	16.0890	22.8702	12.4437	29.6345

Figure 4.23.: Results for system times by student type

4.2. Example: Iterative Looping, Expressions, and Sub-models

In this example, we will work with attributes, variables, and expressions. The model will also introduce modules that facilitate iterative looping (while loops) that you would see in general purpose programming languages. In addition, we will see how Arena facilitates the use of arrays to hold variable values and expressions. The main purpose of this model is to illustrate how to use these programming constructs within an Arena model.

The model introduced in this section 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.
- Sub-models 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. The final model will look something like that shown in Figure 4.24.

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 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 4.1 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.

4. Modeling Systems with Processes and Basic Entity Flow

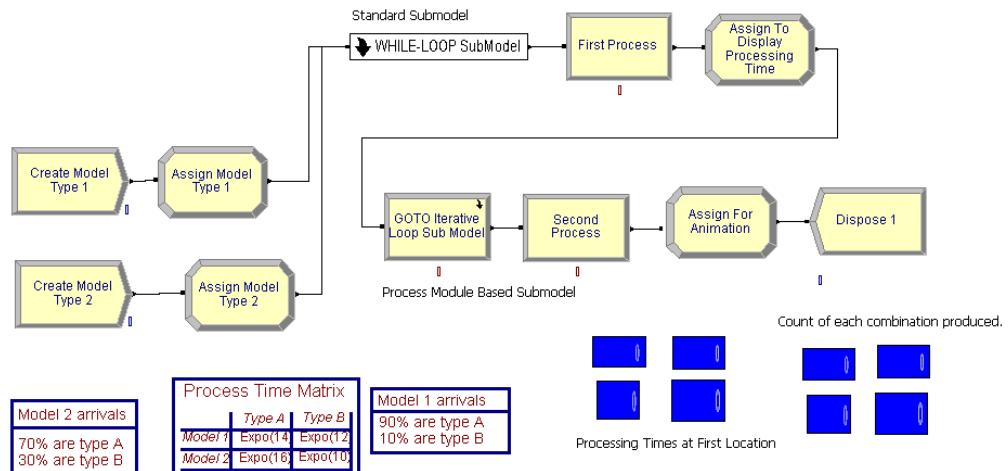


Figure 4.24.: Completed model for iterative looping example

Table 4.1.: Data for the iterative looping example

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

When building this model, we 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.

4.2.0.1. 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 situations, 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. This can be done in two ways 1) by using the ENTITY module to define an entity type or 2) by using user defined attributes. Either method can be used in this situation, but by using user defined attributes, we 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's 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:

- 216 • 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 answer to 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 doesn't 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.

4.2. Example: Iterative Looping, Expressions, and Sub-models

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's 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.

The following pseudo-code for the example is as follows. First, we define the attributes, variables and expressions. For variables, we see something new. We have defined a 2-dimensional variable called vMTCount that will be used to count the number of parts of each model and type combination that are produced. This 2-dimensional variable can be used just like in other programming languages. Similarly, we define a 2-dimensional expression. Just like was illustrated in the last example, an expression can hold any valid Arena expression; however, in this case, we have defined an arrayed expression from which we can look up the appropriate distribution associated with a particular model and product type combination. This is an extremely useful method to store information.

```
// Model Definition Pseudo-Code

ATTRIBUTES:
myType // represents the type of product, 1 = type A, 2 = type B
myModel // represents the type of model, 1 = Model Type 1, 2 = Model Type 2
myProcessingTime // holds the total processing time based on the model/product type

VARIABLES:
vCounter = 0 // used to count the number of iterations in a loop
vNumIterations = 10 // represents the total number of iterations in a loop
vMTCount: 2-D Array, rows = 2 (model type), columns = 2 (product type)
    vMTCount(1,1) = 0, vMTCount(1,2) = 0
    vMTCount(2,1) = 0, vMTCount(2,2) = 0
vPTLocation1: 2-D Array, rows = 2 (model type), columns = 2 (product type)
    vPTLocation1(1,1) = 0, vPTLocation1(1,2) = 0
    vPTLocation1(2,1) = 0, vPTLocation1(2,2) = 0
```

4. Modeling Systems with Processes and Basic Entity Flow

```
vPTLocation2: 2-D Array, rows = 2 (model type), columns = 2 (product type)
vPTLocation2(1,1) = 0, vPTLocation2(1,2) = 0
vPTLocation2(2,1) = 0, vPTLocation2(2,2) = 0
```

EXPRESSIONS:

```
// represents the processing time distribution for model and product type combinations
ePTime: 2-D Array, rows = 2 (model type), columns = 2 (product type)
ePTime(1,1) = EXPO(14), ePTime(1,2) = EXPO(12)
ePTime(2,1) = EXPO(16), ePTime(2,2) = EXPO(10)
```

After defining the constructs to use, we can then outline the processes within pseudo-code. We start with the creation of the two types of model configurations and the assignment of the product types. 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. After the processing is done at the first station, the processing time for the second station is computed. Here the processing time is determined in an iterative fashion 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.

```
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

Label A: determine the processing time for first station
BEGIN ASSIGN //initialize the loop
    vCounter = 1
    vNumIterations = 10
    myProcessingTime = 0
END ASSIGN
WHILE (vCounter <= vNumIterations)
    BEGIN ASSIGN
```

4.2. Example: Iterative Looping, Expressions, and Sub-models

```
// sum up the processing time by model and product type
myProcessingTime = myProcessingTime + ePTime(myModel, myType)
vCounter=vCounter + 1 //increment the counter
END ASSIGN
ENDWHILE
DELAY for myProcessingTime
BEGIN ASSIGN //initialize the loop for second station
vCounter = 0
vNumIterations = 10
myProcessingTime = 0
END ASSIGN
Label B:
BEGIN ASSIGN
// sum up the processing time by model and product type
myProcessingTime = myProcessingTime + ePTime(myModel, myType)
vCounter=vCounter + 1 //increment the counter
END ASSIGN
DECIDE IF (vCounter <= vNumIterations)
  GOTO Label B
END DECIDE
DELAY for myProcessingTime
DISPOSE
```

The model will follow the outline presented within the pseudo-code. To support the modeling within the environment, you must first define a number of variables and other data.

4.2.0.2. VARIABLE Module

Begin by defining the variables to be used in the model. Open the environment and define following variables using the VARIABLE module as shown in Figure 4.25.

- 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 2-dimensional variable (2 rows, 2 columns)
- vPTLocation1 used to assist with displaying the processing time of each model/type combination, a 2-dimensional variable (2 rows, 2 columns)

4. Modeling Systems with Processes and Basic Entity Flow

- vPTLocation2 used to assist with displaying the processing time of each model/type combination, a 2-dimensional variable (2 rows, 2 columns)

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 4.25.: Variable definitions for iterative looping example

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 is considered a mathematical expression. A list of distributions is found in Table E.3 in Appendix E. In the table, the function name is given along with the parameters required by the function. The `stream` parameter indicates an optional parameter which helps in controlling the randomness of the distribution. In addition to using distributions in expressions, you can use logical constructs and mathematical operators to build expressions. Table E.1 of Appendix E provides a list of mathematical functions available. To represent the model/product type distributions, the EXPRESSION data module can be used.

4.2.0.3. 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 wherever the named expression appears in the model. The EXPRESSION module also allows for the modeler to define an array of expressions.

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 4.26.

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 expression. Now that you have the basic data pre-defined 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

4.2. Example: Iterative Looping, Expressions, and Sub-models

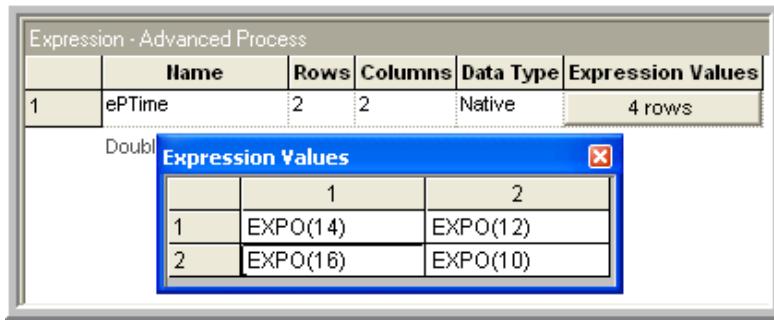


Figure 4.26.: Expression definitions for iterative looping example

should place the two CREATE modules and the two ASSIGN modules into the model window as shown in Figure 4.24. Fill in the create modules as indicated in Figure 4.27. For this example, there will only be 10 products of each model configuration created.

	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 4.27.: Creating the products for iterative looping example

The ASSIGN modules can be used to define the attributes and to 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 4.28.

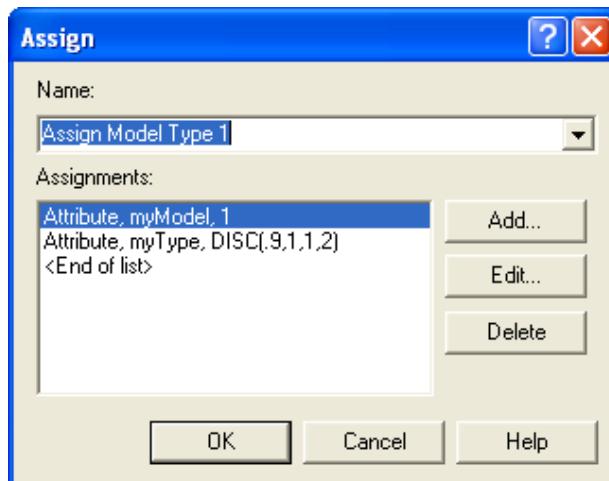


Figure 4.28.: ASSIGN module dialog box for model type 1

4. Modeling Systems with Processes and Basic Entity Flow

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 data sheet view. Click on the assignment rows to get the assignments window and fill it in as indicated in Figure 4.29.

The screenshot shows the 'Assignments' window with two rows of data:

	Type	Attribute Name	New Value
1	Attribute	myModel	2
2	Attribute	myType	DISC(.7,1,1,2)

Below the table, a message says "Double-click here to add a new row."

Below the assignments window, there is a section titled "Assign - Basic Process" containing four rows of data:

	Name	Assignments
1	Assign Model Type 1	2 rows
2	Assign Model Type 2	2 rows
3	Assign For Animation	2 rows
4	Assign To Display Processing Time	1 rows

Figure 4.29.: Second ASSIGN module in data sheet view

4.2.0.4. Hierarchical Sub-models

In this section, the middle portion of the model as indicated in Figure 4.30, which includes a sub-model will be built. A sub-model in is a named collection of modules that is used to organize the main model. Go to the Object menu and choose Sub-model > Add Sub-model. 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 sub-model and selecting the properties menu item, you can give the sub-model a name. This sub-model will demonstrate the use of a WHILE-ENDWHILE loop. 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.

4.2.0.5. WHILE-ENDWHILE Blocks

Open up the sub-model by double clicking on it and add the modules (ASSIGN, WHILE, ASSIGN, and ENDWHILE) to the sub-model as indicated in Figure 4.31. You will find the WHILE and ENDWHILE blocks on the BLOCKS panel. If you haven't already done so 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.

4.2. Example: Iterative Looping, Expressions, and Sub-models

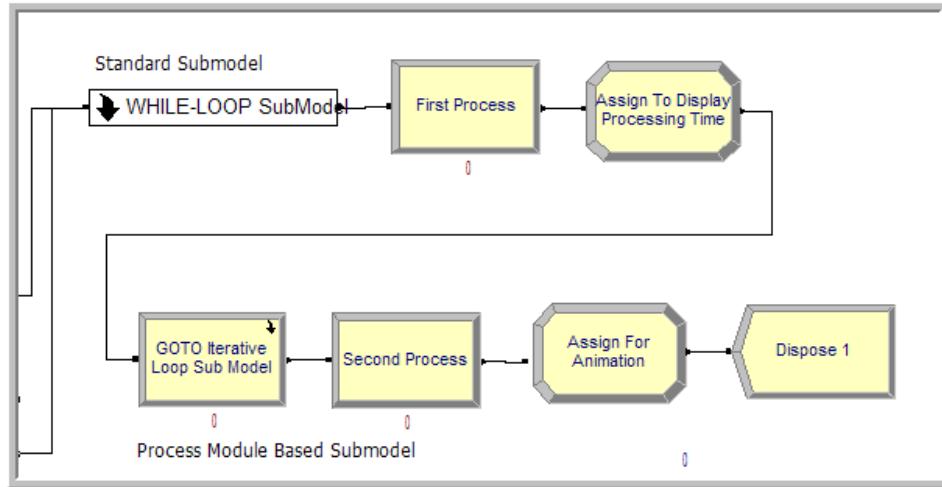


Figure 4.30.: First location processing

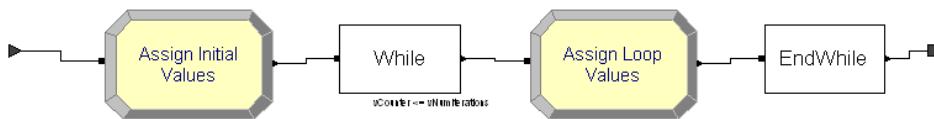


Figure 4.31.: First location processing sub-model

Now we implement the pseudo-code logic within the sub-model. We will 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, HOLD, etc. then it may be possible for another entity to change the value of the counter when the entity in the loop is in the delayed state. This may cause unintended side effects, e.g. 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, 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 4.32. Notice how the variables are initialized prior to the WHILE. In addition, the `myProcessingTime` attribute has been defined and initialized to zero.

4. Modeling Systems with Processes and Basic Entity Flow

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 4.32.: Initializing the looping counter

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 block and make it look like that shown in Figure 4.33. When the variable vCounter is greater than vNumIterations, the WHILE condition will evaluate to false and the entity will exit the WHILE-ENDWHILE loop.

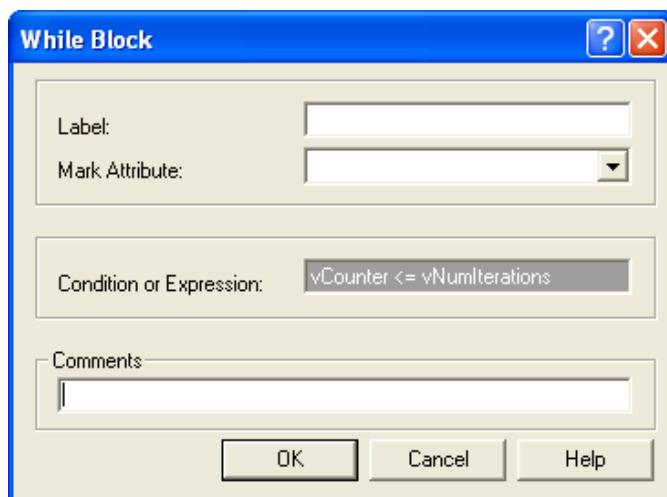


Figure 4.33.: WHILE block within first sub-model

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 4.34. 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 sub-model. Right clicking within the sub-model will give you a context menu for closing the sub-model. After the entity has exited the sub-model, 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.

4.2. Example: Iterative Looping, Expressions, and Sub-models

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 4.34.: Second ASSIGN module with first sub-model

4.2.0.6. PROCESS Module Delay Option

Within 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 that module execution only occurs at event times. moves from event time to event time, executing the associated modules. This concept is illustrated in Figure 4.35 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.

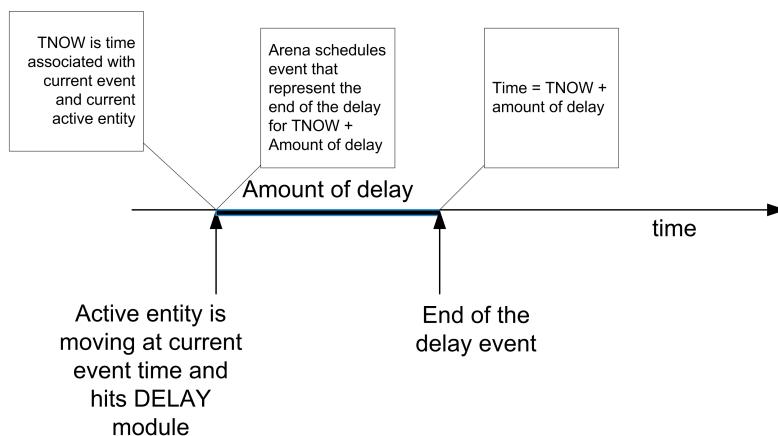


Figure 4.35.: Scheduling a delay

There are two basic ways to cause a delay for an entity, the PROCESS module and the DELAY module. We have already used the delay option of the PROCESS module. The DELAY module is found on the Advanced Process template panel. This example illustrates the use of the DELAY module. The next step in building the model is to place the DELAY module after the sub-model as shown in Figure 4.30. In this case, you want to delay by the amount of time computed for the

4. Modeling Systems with Processes and Basic Entity Flow

processing time. This value is stored in the `myProcessingTime` attribute. Within the expression text field fill in `myProcessingTime` as shown in Figure 4.36. 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.

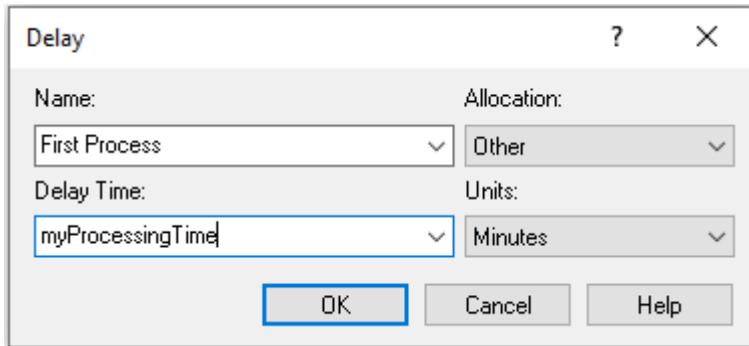


Figure 4.36.: The DELAY module

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 data sheet view, make the ASSIGN module look like that show in Figure 4.37.

Assignments					
	Type	Variable Name	Row	Column	New Value
1	Variable Array (2D)	vPTLocation1	myModel	myType	myProcessingTime

Double-click here to add a new row.

Figure 4.37.: ASSIGN module with arrays

The values of *attributes* cannot be displayed using the variable animation constructs. So, in this ASSIGN module, an array variable is used to temporarily assign 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 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 4.38. 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 4.24 to complete the model. In Figure 4.24, the second sub-model uses a PROCESS based sub-model. It looks like a PROCESS module and is labeled "GOTO Iterative Loop Sub Model", see Figure 4.39. To create this sub-model, lay down a PROCESS module and change its type to sub-model. In what follows, you will examine a different method for looping within the second sub-model. Other than that,

4.2. Example: Iterative Looping, Expressions, and Sub-models

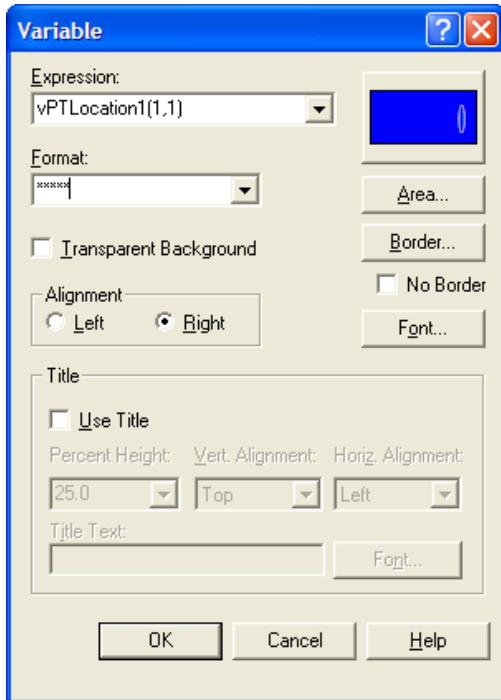


Figure 4.38.: Animate variable dialog box

the rest of the model is similar to what has already been completed. The second sub-model has the modules given in Figure 4.40. Place and connect the ASSIGN, DECIDE, and ASSIGN modules as indicated in Figure 4.41 and Figure 4.42. In 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, e.g. 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 active 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.

4. Modeling Systems with Processes and Basic Entity Flow

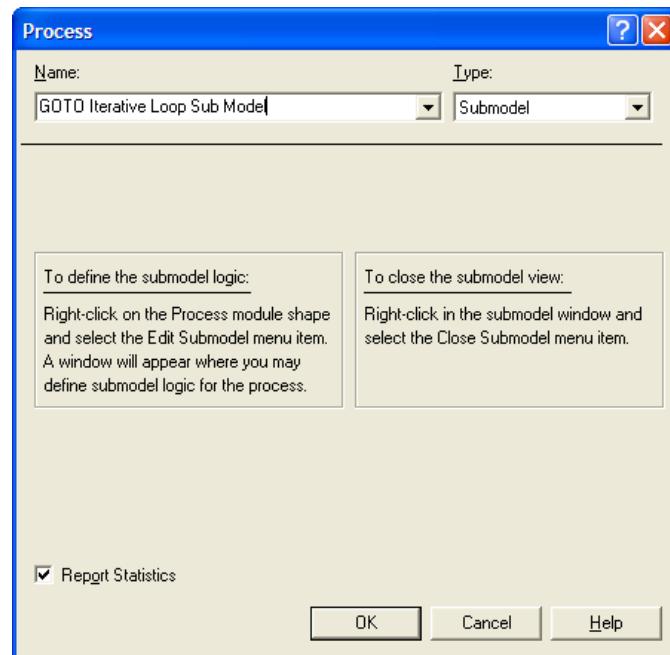


Figure 4.39.: PROCESS based sub-model

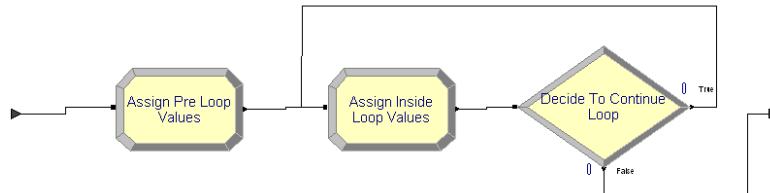


Figure 4.40.: Second sub-model modules

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

Assignments				
	Type	Variable Name	Attribute Name	New Value
1	Attribute	Variable 1	myProcessingTime	myProcessingTime + ePTime(myModel, myType)
2	Variable	vCounter	Attribute 2	vCounter + 1

Figure 4.41.: ASSIGN modules in second sub-model

4.2. Example: Iterative Looping, Expressions, and Sub-models

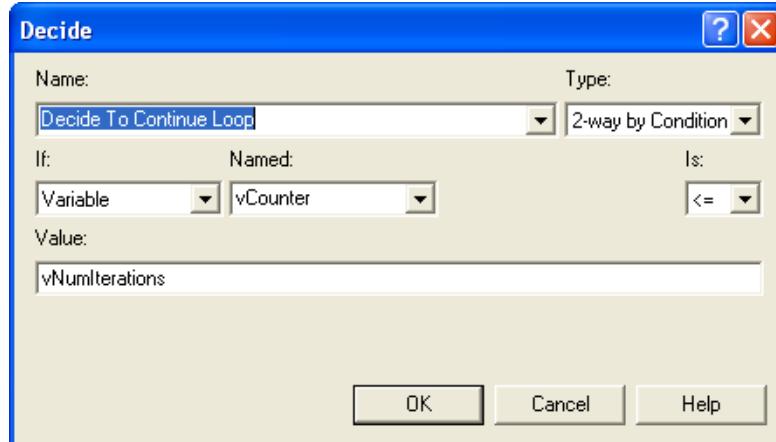


Figure 4.42.: ASSIGN modules in second sub-model

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 module that allows another entity to get to the beginning of the future events list and become the active entity. A blocking module is a module that causes the forward movement of an entity to stop in some manner. There are a variety of modules which will cause this, e.g. HOLD, SEIZE, DELAY, etc. 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 DELAY module is exactly the same as the first DELAY module. In fact, you can just copy the first DELAY module and paste the copy into the model at the appropriate location. If you do copy and paste a module in this fashion, you should make sure to change the name of the module because module names must be unique. The last ASSIGN module is just like the previously explained example. Complete the ASSIGN module as shown in Figure 4.43. Complete the animation of the model in a similar fashion as you did to show the processing time at the first location, except in this case show the counts captured by the variable vMTCount.

Assignments					
	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 4.43.: Last ASSIGN module for animation purposes

To run the model, execute the model using the “VCR” run button. It is useful to use the step

4. Modeling Systems with Processes and Basic Entity Flow

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. The completed model can be found in the files associated with this chapter as file *AttributesVariablesExpressionsLoop-ing.doe*.

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 module was also introduced. Finally, you also refreshed your memory on basic programming constructs such as decision logic and iterative processing.

4.3. Batching and Separating Entities

Of the flow chart 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 5 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 also can *duplicate* or *clone* the incoming entity. This provides an additional mechanism to introduce new entities into the model, i.e. 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.

Example 4.1. 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 hand made to the customer's order design specifications. The time to produce a shirt (of either color) is uniformly distributed within the range of 15 to 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 are 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 to 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.

4.3.1. 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 modeling recipe questions are:

- 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

4. Modeling Systems with Processes and Basic Entity Flow

The entities can be conceptualized as the arriving orders. Since the shirts are processed individually, they should also be considered 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.

The activity diagram for this situation is given in Figure 4.44. 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.

To prepare to represent this situation in pseudo-code, let's define the structural elements of the model.

Pseudo-Code Definition

ATTRIBUTES:

```
myOrderNumber // assigned to provide a unique number to each order  
myOrderType // assigned based on the type of order  
myOrderSize // represents the order size (number of shirts in the order)
```

VARIABLES:

```
vOrderNumber // to count the orders and assign a unique number to each order
```

RESOURCES:

```
Packager (1)  
ShirtMakers (2)
```

The following pseudo-code represents the activity diagram. In the pseudo-code, 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, the order type and the order size are both 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.

```
CREATE orders every hour exponentially  
BEGIN ASSIGN
```

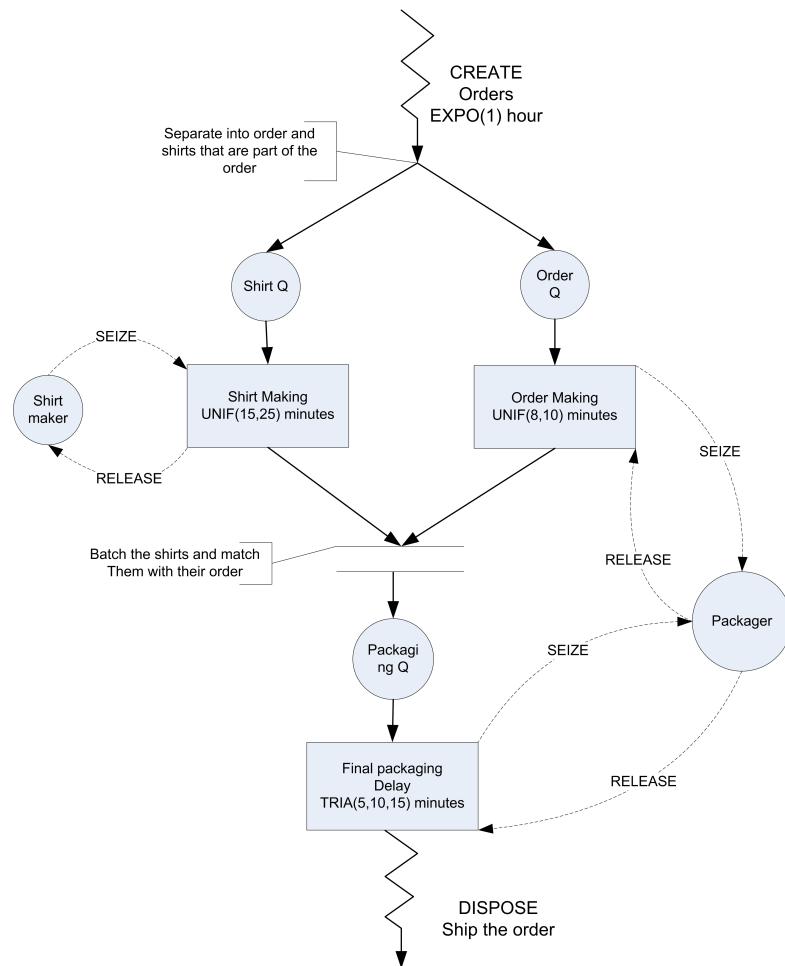


Figure 4.44.: Activity diagram for Tie Dye T-Shirts example

```

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
    Original: Shirts GOTO Label A order making
    Duplicates: Order GOTO Label B shirt making
END SEPARATE
  
```

```

Label A: Order Making
SEIZE packager
DELAY UNIF(8,10) minutes
RELEASE packager
  
```

4. Modeling Systems with Processes and Basic Entity Flow

GOTO Label C, Final Packaging

```

Label B: Shirt Making
SEIZE shirt maker
DELAY UNIF(15,20) minutes
RELEASE shirt maker
BATCH myOrderSize shirts based on myOrderNumber, together
GOTO Label C, Final packaging

```

```

Label C: Final Packaging
BATCH order and shirts together, based on myOrderNumber
SEIZE packager
DELAY TRIA(5,10,15) minutes
RELEASE packager
DISPOSE: Ship order out of the system

```

4.3.2. Building the Model

The completed model is given in Figure 4.45 and can be found in the files associated with this chapter as *TieDyeShirts.doe*. Notice how there is a parallel structure in the middle of the model. This is where the orders are processed separately from the shirts. Parallel processing is quite a common pattern in simulation modeling.

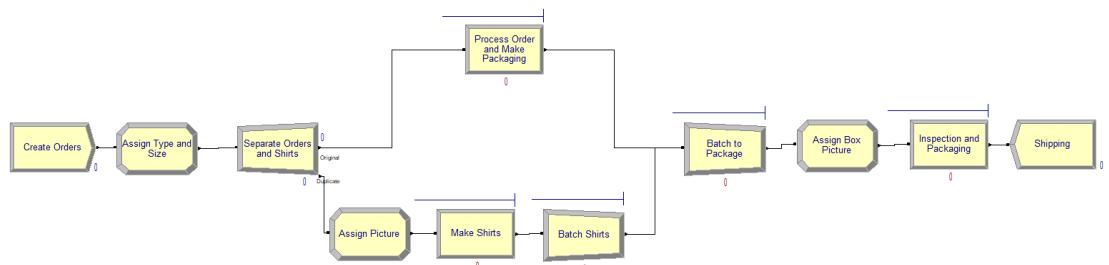


Figure 4.45.: Overall Tie Dye T-Shirt model

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 CREATE module is essentially the same as you have already seen. Open up your CREATE module and fill it out as shown in Figure 4.46.

In the ASSIGN module of Figure 4.47, 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 the given probabilities. Notice the concept of blue and red are mapped to the numbers 1 and 2, respectively. This is often the case within , since attributes and variables can only be real numbers. The `DISC()` distribution is

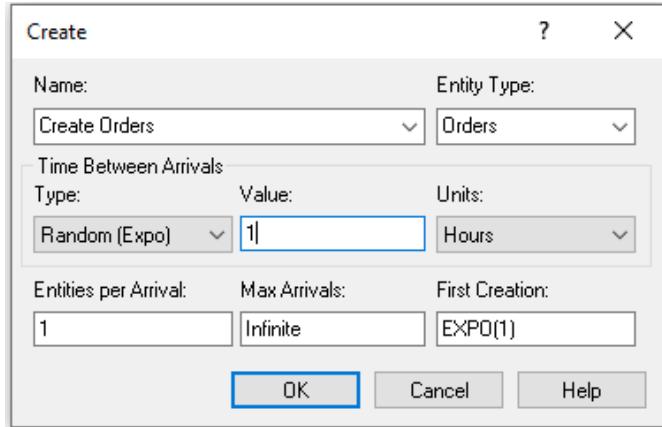


Figure 4.46.: CREATE module for Tie Dye T-Shirt model

used to randomly assign the size of the order. This will be remembered by the entity within the `myOrderSize` attribute.

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 4.47.: Assigning the order number, type, and size

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 4.48. This field indicates how many entities will be created by the SEPARATE module. It can be any valid 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. Don't 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 (`myOrderNumber`) attribute. These

4. Modeling Systems with Processes and Basic Entity Flow

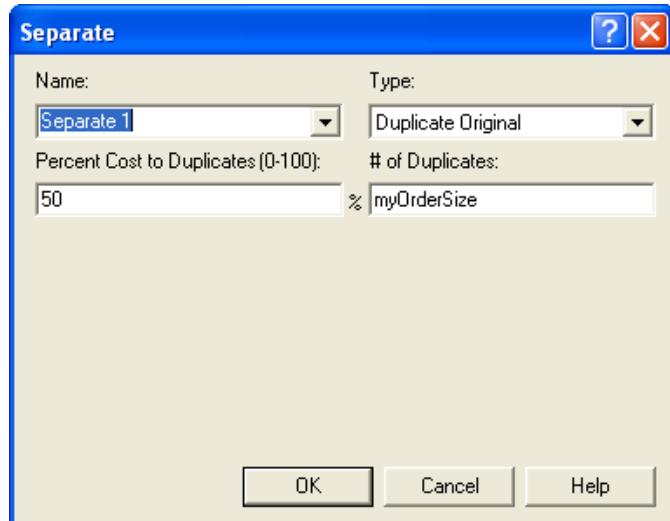


Figure 4.48.: SEPARATE module for creating shirts from orders

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 data sheet view as shown in Figure 4.49. Notice here that there are two units of capacity for the shirt maker resource to represent the two workers involved in this process.

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 4.49.: Defining the resources for Tie Dye T-Shirts example

The packager is used to process the orders along the original entity path. Figure 4.50 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 Figure 4.50 and Figure 4.51.

After the order's packaging is complete, it must wait until the shirts associated with the order are made. As seen in Figure 4.45, the orders go to a BATCH module, where they will be batched by the attribute `myOrderNumber` with the associated batch 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 4.52. This BATCH module creates a permanent entity because the shirts do not need to be process individually after being combined into the size of the order. The entities

4.3. Batching and Separating Entities

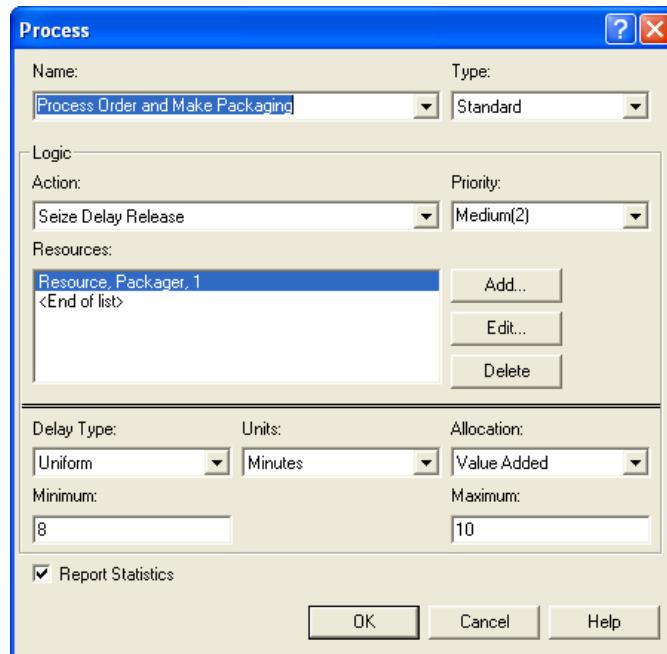


Figure 4.50.: Seizing, delaying, and releasing the packager

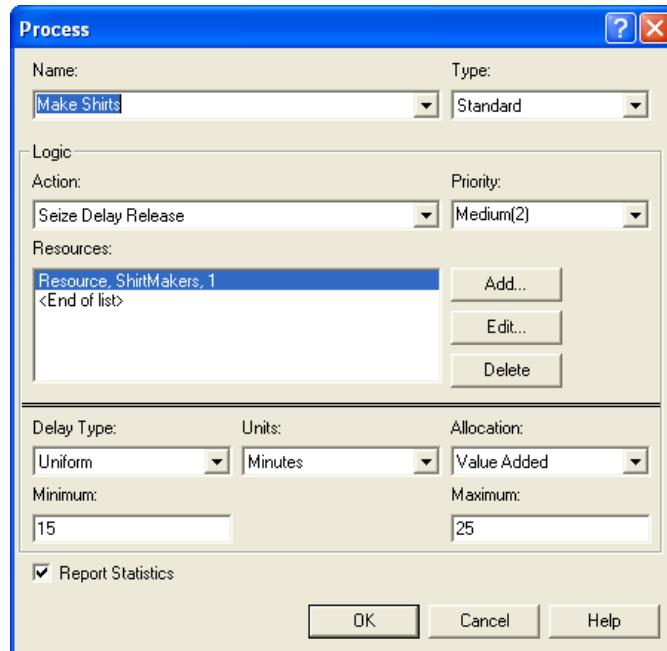


Figure 4.51.: Seizing, delaying, and releasing the shirt makers

4. Modeling Systems with Processes and Basic Entity Flow

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.

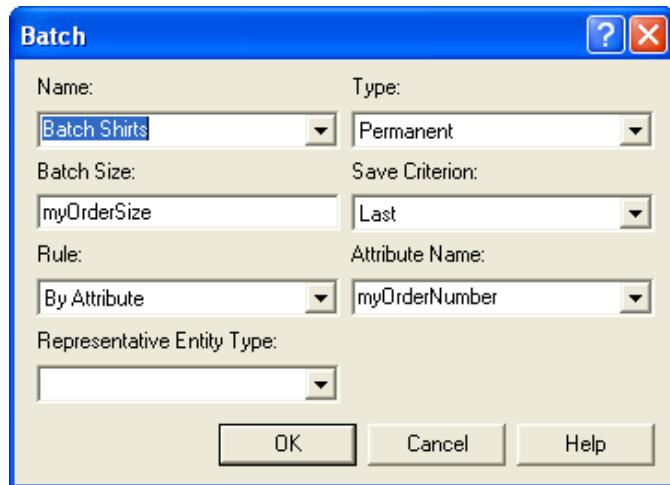


Figure 4.52.: Batching the shirts in an order together

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.

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.

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 second BATCH module (Figure 4.53) where they will be combined with their associated order. 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 4.54) where it is inspected and then sent to a DISPOSE module, which represents shipping.

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

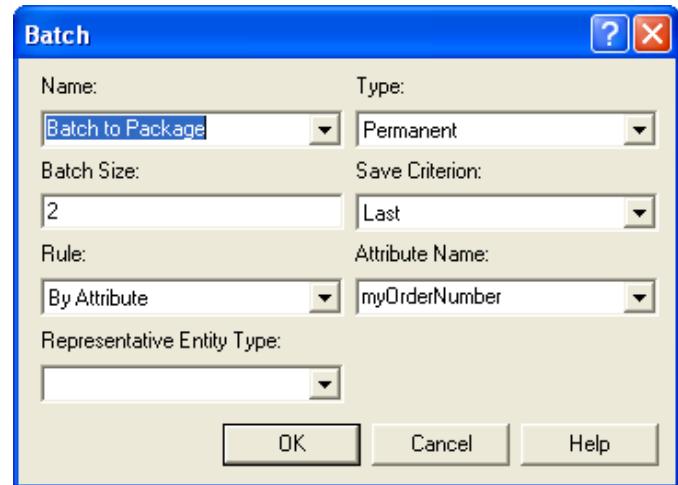


Figure 4.53.: Batching the shirts and the order together

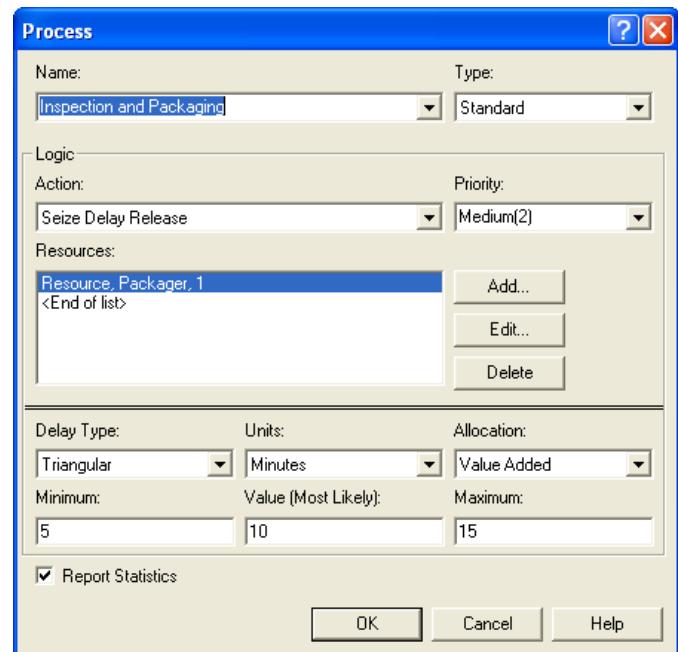


Figure 4.54.: Using the packager to do inspection and packaging

4. Modeling Systems with Processes and Basic Entity Flow

of the SEPARATE 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 . In this instance, a set is 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 4.55. 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.

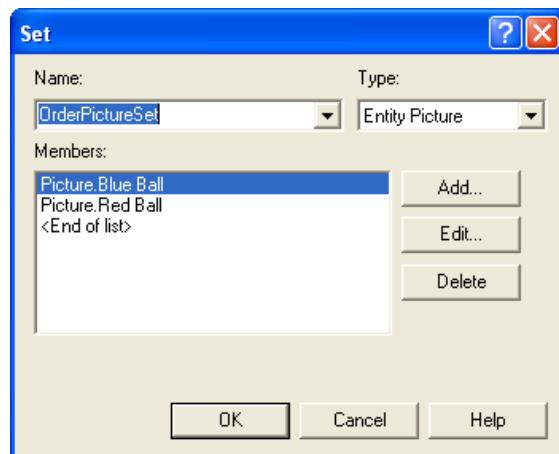


Figure 4.55.: Defining a picture set

In Figure 4.57, an assignment of type “Other” has been made to the Entity.Picture special purpose attribute. The expression builder was used to build the appropriate expression for indexing into the set as shown in Figure 4.56. The resulting ASSIGN module is shown in Figure 4.57. Figure 4.58 indicates the ASSIGN module for directly assigning a 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 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 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. Finally, you saw that that also has another data structure for holding information (besides variables and attributes) called a set. A set acts as a

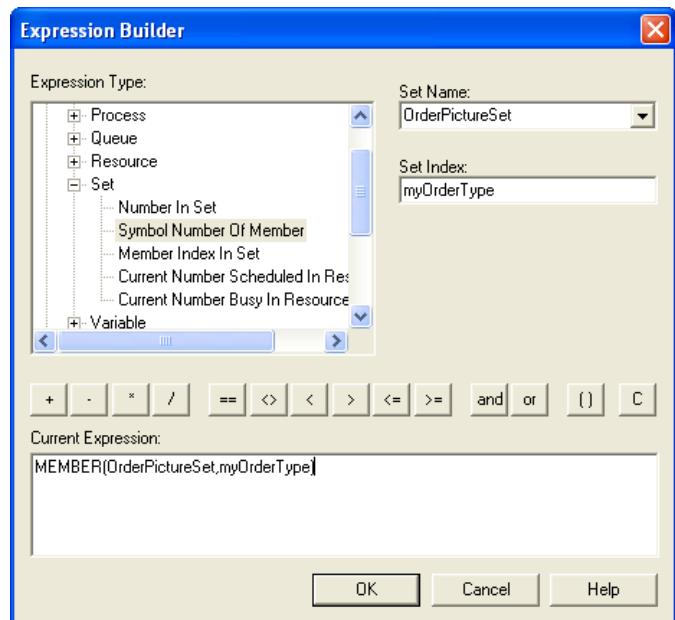


Figure 4.56.: Using the expression builder to index into a set

	Type	Other	New Value
1	Other	Entity.Picture	MEMBER(OrderPictureSet,myOrderType)

Double-click here to add a new row.

Figure 4.57.: Assigning an animation picture based on a set index

	Type	Entity Picture
1	Entity	Picture.Box

Double-click here to add a new row.

Figure 4.58.: Directly assigning an animation picture

4. Modeling Systems with Processes and Basic Entity Flow

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.



What is the difference between a DECIDE module and a SEPARATE module? It seems like DECIDE and SEPARATE control the flow of the entities. There are two output ports on each of the modules. However, these modules function quite differently. The DECIDE module allows for decision-making processes in the model. One entity enters and the *same* entity exits on one of the output ports. For the SEPARATE module, one entity enters and potentially *multiple* entities exit. The original entity exits the port labeled “original” and duplicates exit the port labeled “duplicates”. A SEPARATE module can also be used to split an entity that has been batched together. The CLONE module also has similar functionality.

In the next section, we will review some statistical concepts related to comparing two random samples. Then, we will examine another modeling situation that uses the SEPARATE module and then use that model to compare two design configurations.

4.4. Statistical Issues When Comparing Two Systems

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, service distribution, etc.) and the structure of the model (e.g. FIFO queue, process flow, etc.) 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 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 4.59 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, 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.

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)).

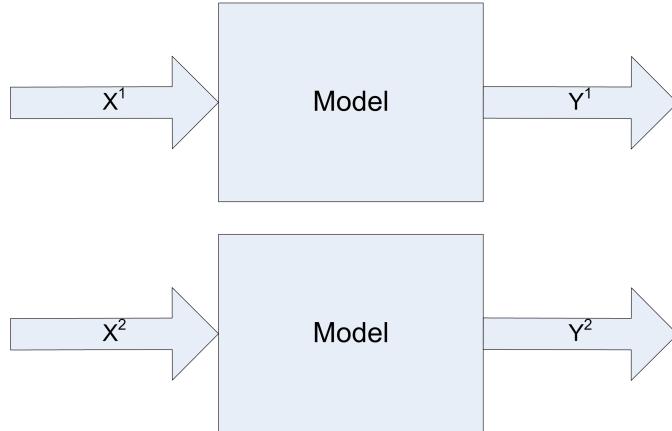


Figure 4.59.: Changing inputs on models represent different system configurations

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

4. Modeling Systems with Processes and Basic Entity Flow

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 (re-use) 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 presents 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 to do the work of the analysis will be illustrated.

4.4.1. 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 ($k = 1, 2$):

$$\bar{X}_k = \frac{1}{n_k} \sum_{j=1}^{n_k} X_{kj}$$
$$S_k^2 = \frac{1}{n_k - 1} \sum_{j=1}^{n_k} (X_{kj} - \bar{X}_k)^2$$

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:

4.4. Statistical Issues When Comparing Two Systems

$$Var(\hat{D}) = Var(\bar{X}_1 - \bar{X}_2) = \frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}$$

where σ_1^2 and σ_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:

$$Var(\hat{D}) = Var(\bar{X}_1 - \bar{X}_2) = \frac{\sigma^2}{n_1} + \frac{\sigma^2}{n_2} = \sigma^2 \left(\frac{1}{n_1} + \frac{1}{n_2} \right)$$

where σ^2 is the common unknown variance. A pooled estimator of σ^2 can be defined as:

$$S_p^2 = \frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2}$$

Thus, a $(1 - \alpha)\%$ confidence interval on $\theta = \theta_1 - \theta_2$ is:

$$\hat{D} \pm t_{1-(\alpha/2),v} s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

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:

$$\hat{D} \pm t_{1-(\alpha/2),v} \sqrt{S_1^2/n_1 + S_2^2/n_2}$$

where

$$v = \left\lfloor \frac{\left(\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2} \right)^2}{\frac{(S_1^2/n_1)^2}{n_1+1} + \frac{(S_2^2/n_2)^2}{n_2+1}} - 2 \right\rfloor$$

Let $[l, u]$ be the resulting confidence interval where l and u represent the lower and upper limits of the interval with by construction $l < u$. Thus, if $u < 0$, you can conclude with $(1 - \alpha)\%$ confidence that $\theta = \theta_1 - \theta_2 < 0$ (i.e. that $\theta_1 < \theta_2$). If $l > 0$, you can conclude with $(1 - \alpha)\%$ that $\theta = \theta_1 - \theta_2 > 0$ (i.e. that $\theta_1 > \theta_2$). 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.

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 Table 4.2.

4. Modeling Systems with Processes and Basic Entity Flow

Table 4.2.: Comparing two designs

	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 as F.INV.RT(p, u, v) within Excel. Note also that $f_{1-p, u, v} = 1/f_{p, v, u}$. In R, the formula is $f_{p, u, v} = qf(1-p, u, v)$, 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:

$$\begin{aligned}\hat{\theta} &\pm t_{1-(\alpha/2),v} s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}} \\ 0.66 &\pm t_{1-0.025,20} (\sqrt{0.384}) \sqrt{\frac{1}{10} + \frac{1}{12}} \\ 0.66 &\pm (2.086)(0.6196)(0.428) \\ 0.66 &\pm 0.553\end{aligned}$$

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 4.60 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).

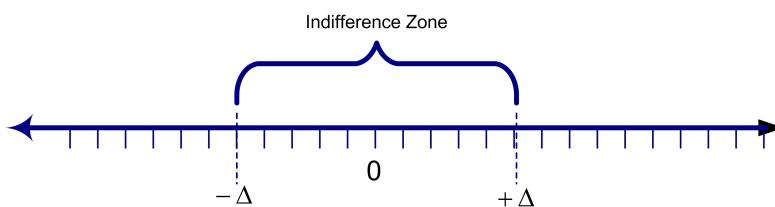


Figure 4.60.: Concept of an indifference zone

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

4. Modeling Systems with Processes and Basic Entity Flow

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.

4.4.2. 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 the $(X_{11}, X_{12}, \dots, X_{1n_1})$ 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 re-executed 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, i.e. $n_1 = n_2 = n$. Since each replication for the two systems uses the same random number streams, the correlation between $(X_{1,j}, X_{2,j})$ will not be zero; however, each pair will still be independent *across* the replications. The basic approach to analyzing this situation is to compute the difference for each pair:

$$D_j = X_{1j} - X_{2j} \text{ for } j = 1, 2, \dots, n$$

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:

$$\begin{aligned}\bar{D} &= \frac{1}{n} \sum_{j=1}^n D_j \\ S_D^2 &= \frac{1}{n-1} \sum_{j=1}^n (D_j - \bar{D})^2 \\ \bar{D} \pm t_{1-\alpha/2, n-1} \frac{S_D}{\sqrt{n}}\end{aligned}$$

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.

Assume that the two simulations were run independently using common random numbers. Recommend the better design with 95% confidence.

4.4. Statistical Issues When Comparing Two Systems

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:

$$\begin{aligned}\hat{D} &\pm t_{1-(\alpha/2), n-1} \frac{S_D}{\sqrt{n}} \\ 2.22 &\pm t_{1-0.025, 9} \frac{0.55}{\sqrt{10}} \\ 2.22 &\pm (2.261)(0.1739) \\ 2.22 &\pm 0.0.393\end{aligned}$$

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 versus 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 affect on the variance of the estimator. In the case of independent samples, the estimator of performance was $\bar{D} = \bar{X}_1 - \bar{X}_2$. Since

$$\begin{aligned}\bar{D} &= \frac{1}{n} \sum_{j=1}^n D_j \\ &= \frac{1}{n} \sum_{j=1}^n (X_{1j} - X_{2j}) \\ &= \frac{1}{n} \sum_{j=1}^n X_{1j} - \frac{1}{n} \sum_{j=1}^n X_{2j} \\ &= \bar{X}_1 - \bar{X}_2 \\ &= \hat{D}\end{aligned}$$

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:

$$V_{\text{IND}} = \text{Var}(\bar{X}_1 - \bar{X}_2) = \frac{\sigma_1^2}{n} + \frac{\sigma_2^2}{n}$$

Under the assumption that the samples are not independent, the variance of the estimator is:

4. Modeling Systems with Processes and Basic Entity Flow

$$V_{\text{CRN}} = \text{Var}(\bar{X}_1 - \bar{X}_2) = \frac{\sigma_1^2}{n} + \frac{\sigma_2^2}{n} - 2\text{cov}(\bar{X}_1, \bar{X}_2)$$

If you define $\rho_{12} = \text{corr}(\bar{X}_1, \bar{X}_2)$, the variance for the common random number situation is:

$$V_{\text{CRN}} = V_{\text{IND}} - 2\sigma_1\sigma_2\rho_{12}$$

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 θ 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.



Use common random numbers when comparing two alternatives to block out the factor of the assigned random numbers and reduce the variance of your estimates for the difference between the system configurations.

In experimental design settings, this is called blocking on a factor. For example, if you wanted to perform an 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 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.

In the next section, we introduce a modeling situation involving the production of rings. We will then apply the methods of this section to compare two design configurations involving this system.

4.5. The LOTR Makers, Inc. Example

In this section, we will develop a model for a small manufacturing system. The purpose of the model is to illustrate how to model with a variety of different distributions, illustrate a common entity creation pattern, and cover the use of resource sets. In a future section we will build on this model in order to compare two system design configurations in a statistically valid manner. We start by introducing the modeling situation. The situation is fictitious but has a number of interesting modeling issues.

Example 4.2. 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 to 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 cm 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 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 cm.

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 log-normal 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

4. Modeling Systems with Processes and Basic Entity Flow

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.

4.5.1. Conceptualizing the Model

Now let's proceed with the modeling of Example 4.2. 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 this:

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 this:

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 $p \sim \text{BETA}(\alpha_1 = 5, \alpha_2 = 1.5)$. 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 the following 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.

4. Modeling Systems with Processes and Basic Entity Flow

```

CREATE 1 order process logical entity
BEGIN ASSIGN
    vSalesProb = BETA (5, 1.5)
    myCallNum = 1
END ASSIGN
WHILE myCallNum <= 100
    BEGINN ASSIGN
        myCallNum = myCallNum + 1
        mySale = DISC(vSalesProb,1,1.0,0)
        myNumSales = myNumSales + mySale
    END ASSIGN
ENDWHILE
SEPARATE
    Original: dispose of original order logical entity
    Duplicate: create myNumSales duplicates, send to LABEL Production
END SEPARATE

```

The production process pseudo-code describes what happens to a pair of rings as it moves through production.

```

LABEL: Production
BEGIN PROCESS Ring Making
    SEIZE master ring maker
    DELAY for ring making
    RELEASE master ring maker
END PROCESS
BEGIN ASSIGN
    myIDBigRing = NORM(1.49, 0.005)
    myODSmallRing = NORM(1.5, 0.002)
    myGap = myIDBigRing - myODSmallRing
END ASSIGN
BEGIN PROCESS Inspection
    SEIZE inspector
    DELAY for inspection
    RELEASE inspector
END PROCESS
DECIDE IF myODSmallRing > myIDBigRing
    RECORD too big
    GOTO LABEL: Rework
ELSE
    RECORD not too big
    IF myGap > 0.02
        RECORD too small

```

```

GOTO LABEL: Rework
ELSE
    RECORD not too small
ENDIF
END DECIDE
LABEL: Packaging
BEGIN PROCESS Packaging
    SEIZE packager
    DELAY for packaging
    RELEASE packager
END PROCESS
DISPOSE

LABEL: REWORK
BEGIN PROCESS Rework
    SEIZE rework craftsman
    DELAY for rework
    RELEASE rework craftsman
END PROCESS
GOTO LABEL: Packaging

```

4.5.2. 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. 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 1 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 section 4.2. An overview of the sales order portion of the model is given in Figure 4.61. 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 1 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

4. Modeling Systems with Processes and Basic Entity Flow

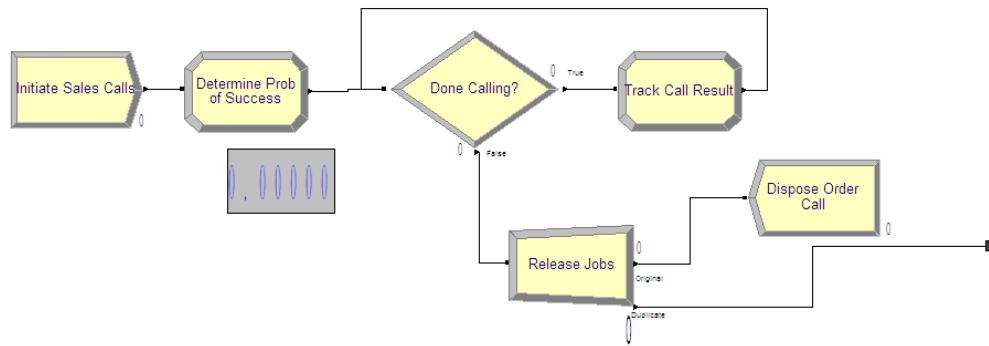


Figure 4.61.: Sales confirmation process

arrivals generated from this CREATE module. Fill out your CREATE module as shown in Figure 4.62.

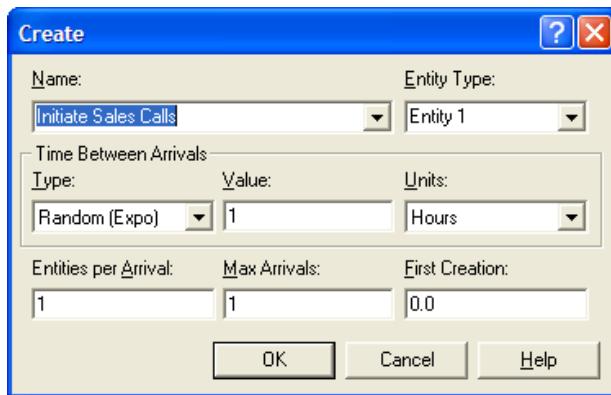


Figure 4.62.: Initiating the sales call confirmation process

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(5,1.5)` distribution function (Figure 4.63). The attribute (`myCallNum`) is initialized to 1, prior to starting the looping logic. This attribute is going to count the number of calls made.

Assignments				
	Type	Variable Name	Attribute Name	
1	Variable	<code>vSalesProb</code>	<code>myOrderProb</code>	<code>BETA(5,1.5)</code>
2	Attribute	<code>Variable 2</code>	<code>myCallNum</code>	1

Figure 4.63.: Determining the probability of success

The DECIDE module in Figure 4.64 uses the attribute (`myCallNum`) to check the variable `vNumCalls`. The variable (`vNumCalls`) is defined in the VARIABLE module (not shown here) and has been ini-

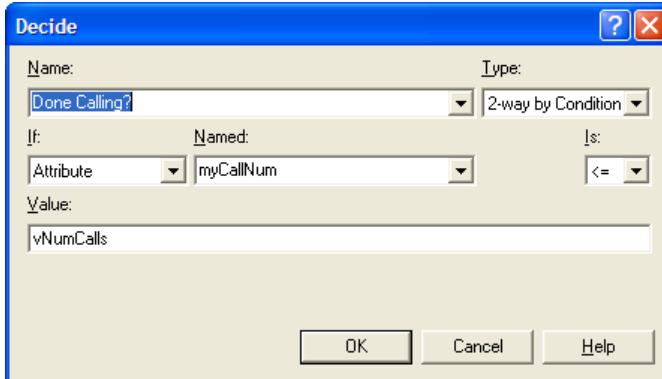


Figure 4.64.: Checking that all calls have been made

tialized 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.

Assignments				
	Type	Attribute Name	New Value	
1	Attribute	myCallNum	myCallNum + 1	
2	Attribute	mySale	DISC(vSalesProb, 1, 1, 0, 0)	
3	Attribute	myNumSales	myNumSales + mySale	
Double-click here to add a new row.				

Figure 4.65.: Tracking the call results

Figure 4.65 presents the ASSIGN module in the looping logic. In this ASSIGN, 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 a 1).

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 4.66). The original order process logic entity is sent to a DISPOSE, 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

4. Modeling Systems with Processes and Basic Entity Flow

logic entity, this is advisable.

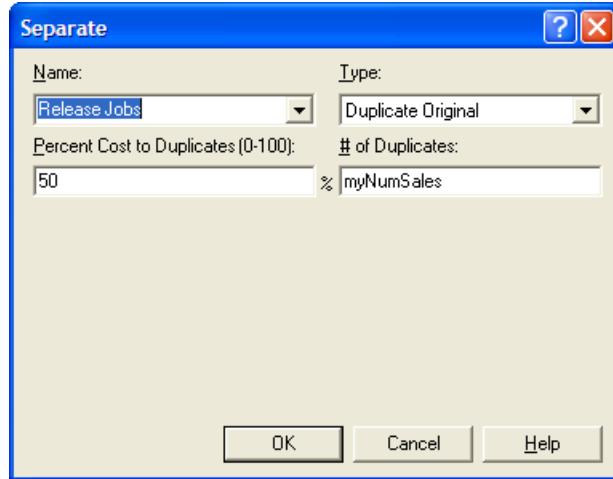


Figure 4.66.: Creating the jobs for production

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 4.67 presents the making and inspection processes which are implemented with PROCESS modules using the SEIZE, DELAY, RELEASE option. Figure 4.68 and Figure 4.69 show the PROCESS modules and the appropriate delay distributions using the uniform and triangular distributions.

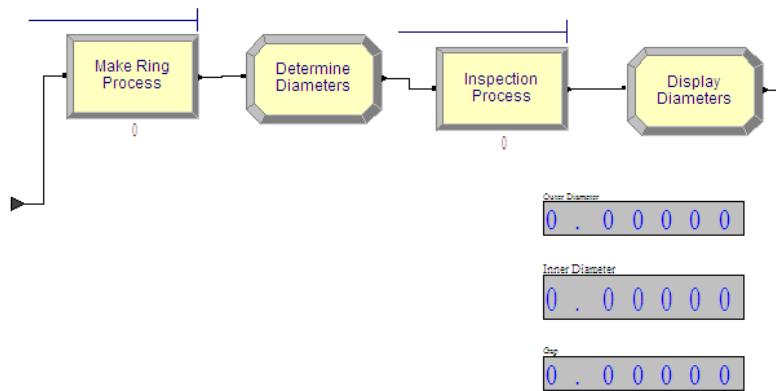


Figure 4.67.: Making and inspecting the rings

The ASSIGN module between the two processes is used to determine the diameters of the rings. Figure 4.70 shows that attributes (`myIDBigRing` and `myODSmallRing`) are both set using the normal distribution. The parameters of the `NORM()` functions are defined by variables in the VARIABLE module (not shown here).

4.5. The LOTR Makers, Inc. Example

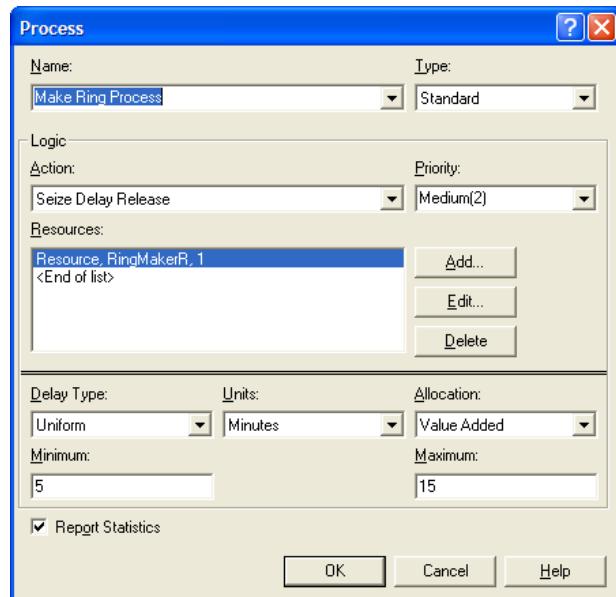


Figure 4.68.: PROCESS module for making the rings

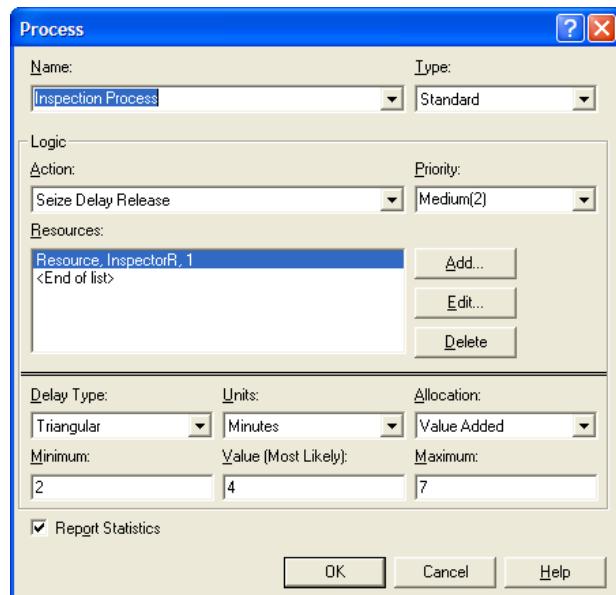


Figure 4.69.: PROCESS module for inspecting the rings

4. Modeling Systems with Processes and Basic Entity Flow

	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 4.70.: Determining the ring diameters

After inspection, the rings must be checked to see whether rework is necessary. An overview of this checking is given in Figure 4.70. 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.

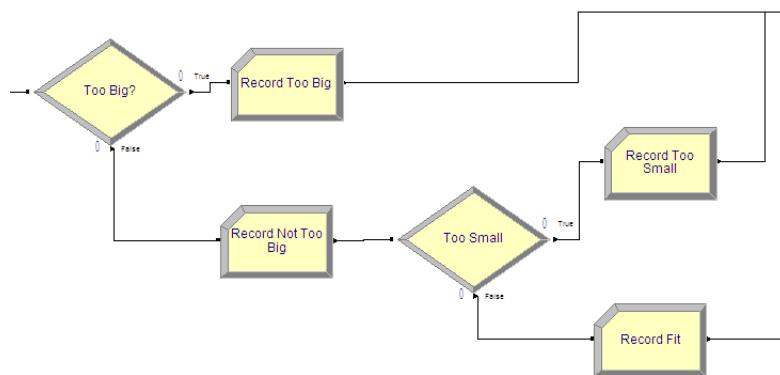


Figure 4.71.: Checking ring diameters

Figure 4.72 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 4.73 shows the DECIDE module for checking if the smaller ring is too loose. In this DECIDE module, the two-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 4.74 shows the rework and packaging processes. Again, the PROCESS module is used to represent these processes.

Figures 4.75 and Figure 4.76 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.

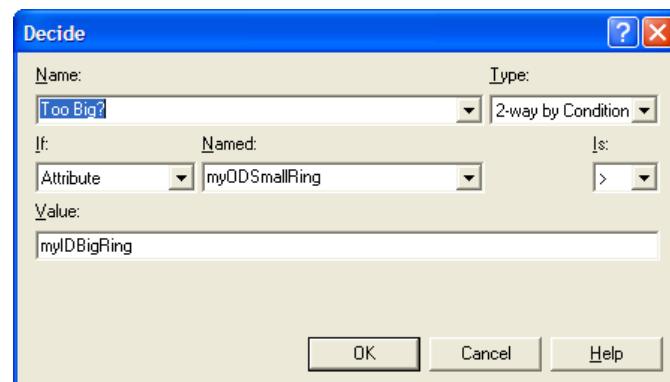


Figure 4.72.: Checking to determine whether small ring is too big

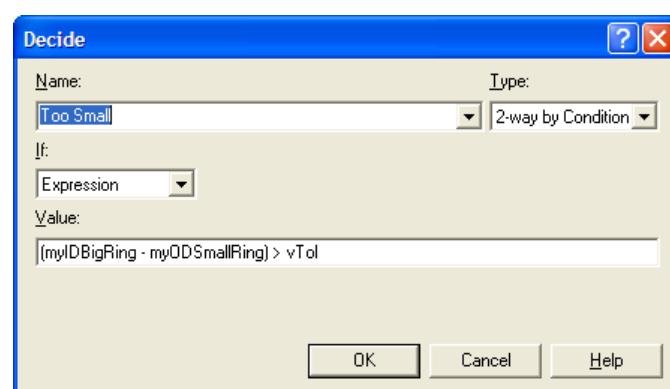


Figure 4.73.: Checking to determine whether small ring is too small

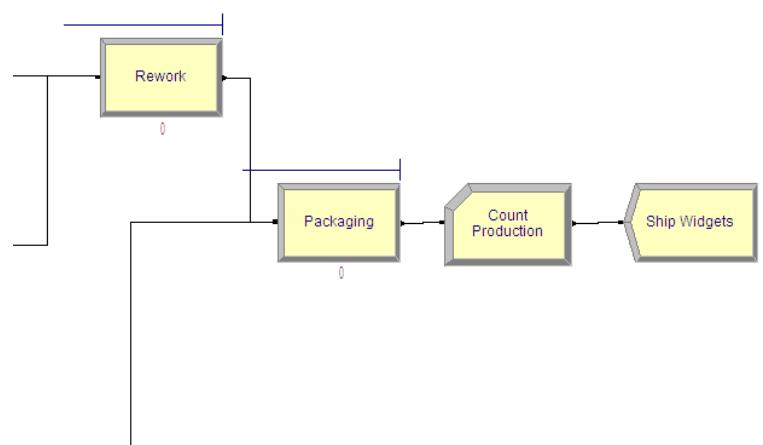


Figure 4.74.: Rework and packaging processes

4. Modeling Systems with Processes and Basic Entity Flow

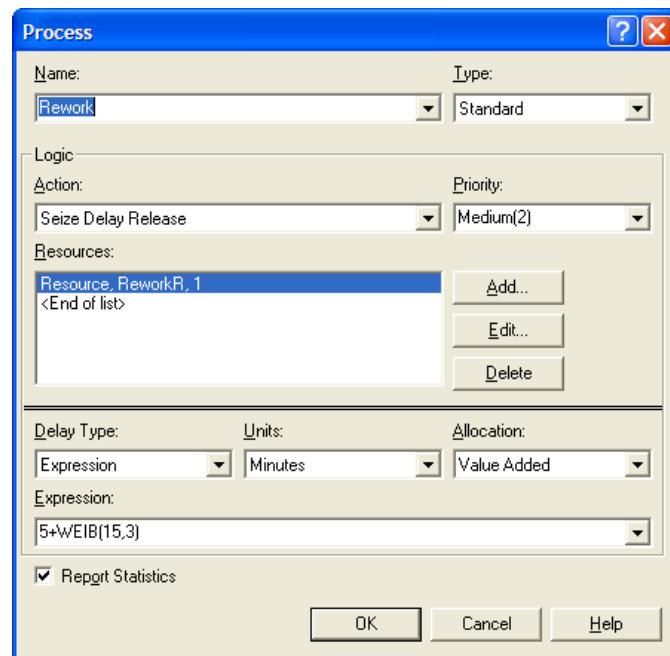


Figure 4.75.: Rework PROCESS module

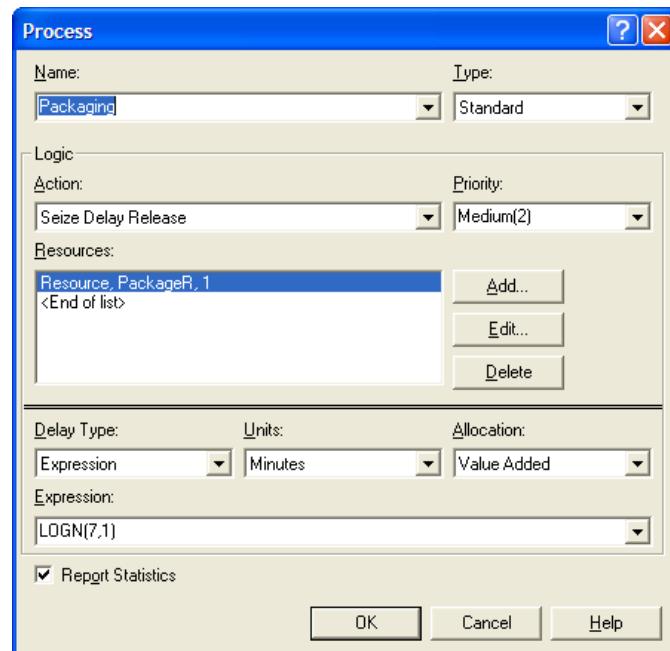


Figure 4.76.: Packaging PROCESS module

4.5.3. Running the Model

The completed model for Example 4.2 can be found in the chapter files as *LOTRExample.doe*. In this section, we will setup and run the model. 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 package the rings, the time that the simulation will end is a random variable. If this time is less than 960 (the time of two shifts), there will not be any overtime. If this time is greater than 960, 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.

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 4.77, 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 4.77 defines an OUTPUT statistic called `ProbOvOT` 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 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

4. Modeling Systems with Processes and Basic Entity Flow

	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 4.77.: Defining OUTPUT statistics for overtime

is an estimate of the probability of the condition. Thus, you will get an estimate of the likelihood of overtime.

Figure 4.78 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 nor is there a terminating condition specified. As previously mentioned, the simulation end 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.

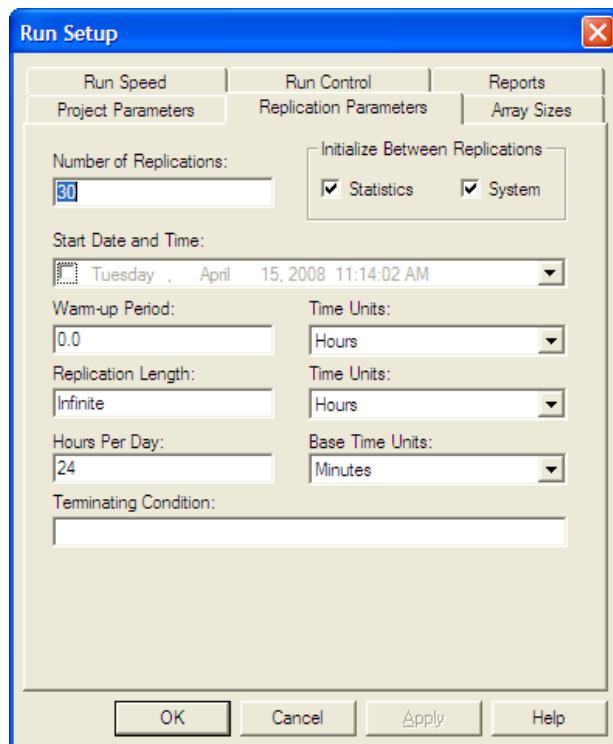


Figure 4.78.: Specifying the number of replications in run setup

4.6. Comparing Two Alternative Configurations for the LOTR Makers

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 4.79. 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, you are asked to explore the reasons behind the overtime and to recommend an alternative to reduce the likelihood of overtime.

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 4.79.: LOTR model results across 30 replicated days

In this example, you have learned how to model a small system involving random components and to translate the conceptual model into an simulation. 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.

4.6. Comparing Two Alternative Configurations for the LOTR Makers

Suppose LOTR Makers 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 give 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 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

4. Modeling Systems with Processes and Basic Entity Flow

must travel to the ring station for the making of the rings. The process takes 12 minutes on 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 4.80 illustrates the two system configurations in the form of activity diagrams. Configuration two 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.

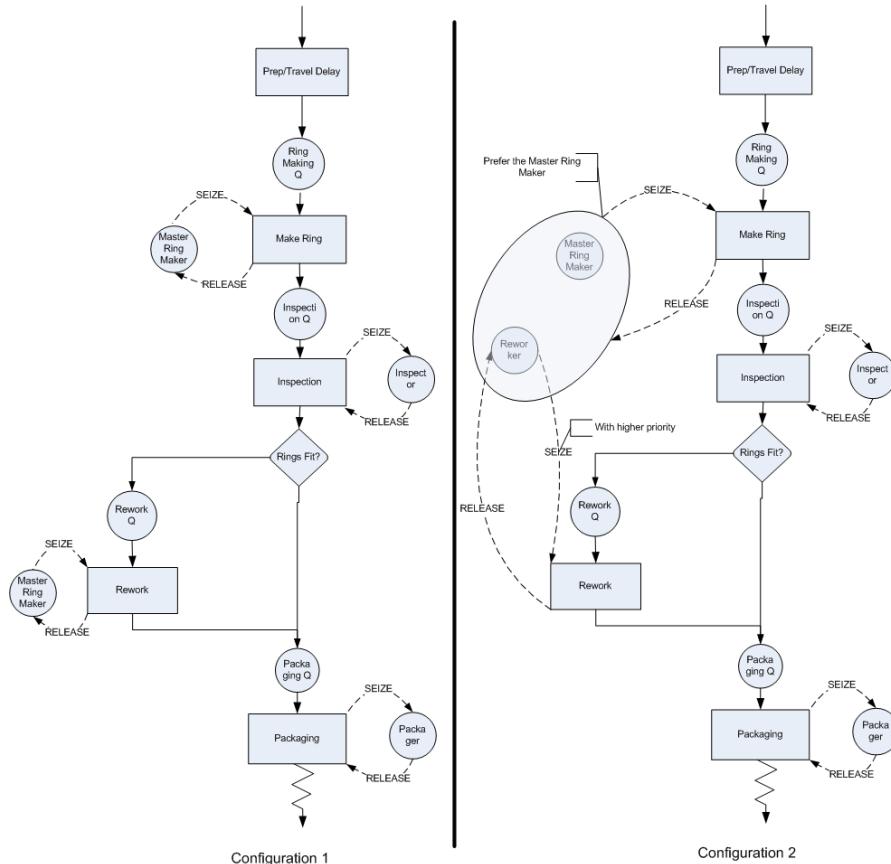


Figure 4.80.: Two LOTR alternative configurations

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 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.

4.6.1. Resource Sets

The diagram indicates that the master craftsman and the rework craftsman are within the same set. A set construct is used to hold various objects of the same type (e.g. resources, queues, etc.). 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 2nd 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.

4. Modeling Systems with Processes and Basic Entity Flow

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 7 default rules and 2 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.

SRC Select 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 rule User Rule defined in the experiment frame.

UR(User Rule) Selects the URth 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.

4.6. Comparing Two Alternative Configurations for the LOTR Makers

Now, you are ready to implement this situation by modifying the file, *LOTREExample.doe*. The files related to this section are available in the book support files for this chapter in a folder called, *ComparingTwoSystemsCRN*. Open up the submodel named, *Ring Processing*, and insert a **DELAY** module at the beginning of the process as shown in Figure 4.81. 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, *LOTRConfig1.doe*, to represent the first system configuration. You will now edit this model to create the second system configuration.

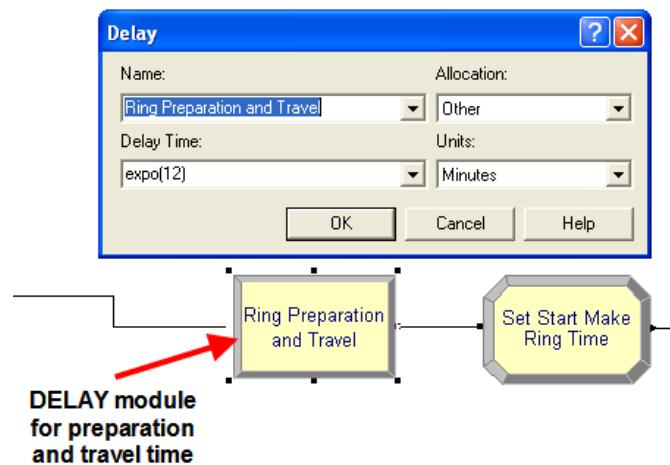


Figure 4.81.: DELAY module for preparation and travel time

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 4.82. Now, you can click on the **Members** area to add rows for each member. Do this as shown in Figure 4.82 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 textbox 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 *LOTRConfig2.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 4.83, 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 4.83, there is a text field labeled **Save Attribute**. This attribute will

4. Modeling Systems with Processes and Basic Entity Flow



Figure 4.82.: Adding members to a resource set

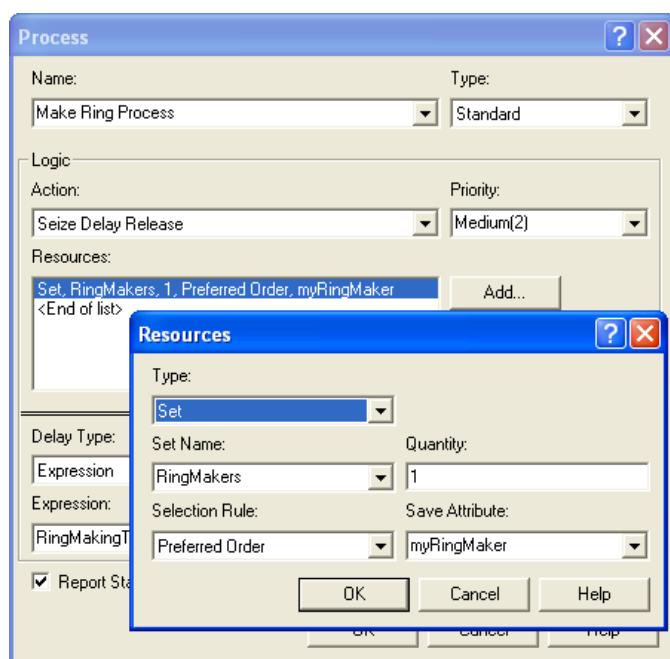


Figure 4.83.: Using a resource set in a PROCESS module

4.6. Comparing Two Alternative Configurations for the LOTR Makers

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.

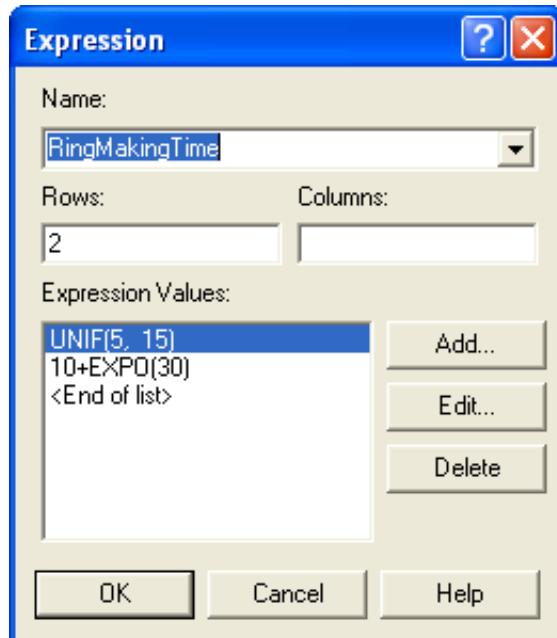


Figure 4.84.: Expressions for ring making time by type of worker

Since there are only two ring makers, an arrayed EXPRESSION can be defined, see Figure 4.84, 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 4.84, open up your Make Ring Process module and edit it according to Figure 4.85. 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 4.86. 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. Go to the STATISTICS module and add a file name (`prodTimeC2.dat`) to the OUTPUT statistic for the time to make the order statistics as shown in Figure 4.87. 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

4. Modeling Systems with Processes and Basic Entity Flow

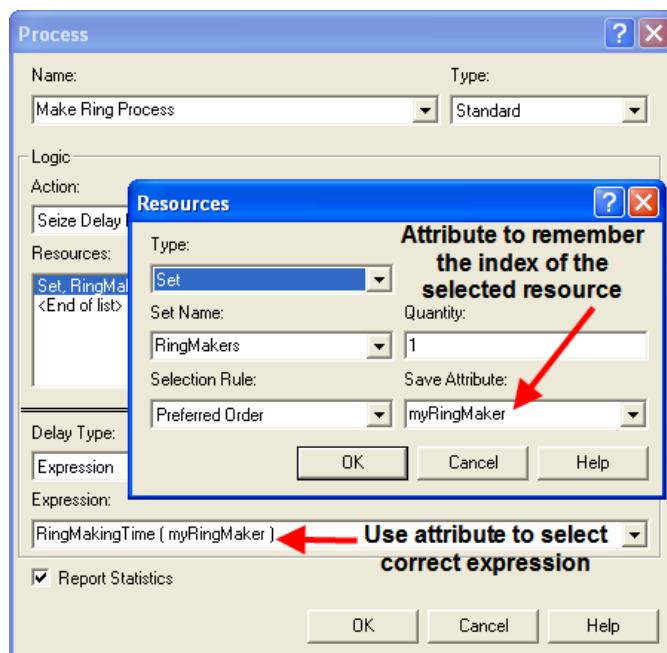


Figure 4.85.: Using the save attribute to remember the selected resource and index into the expression

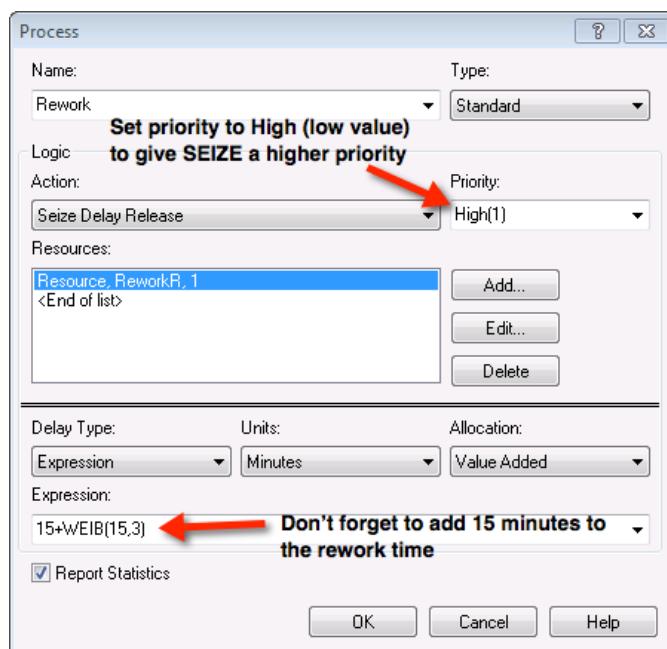


Figure 4.86.: Adjusting the priority when seizing the resource

4.6. Comparing Two Alternative Configurations for the LOTR Makers

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.

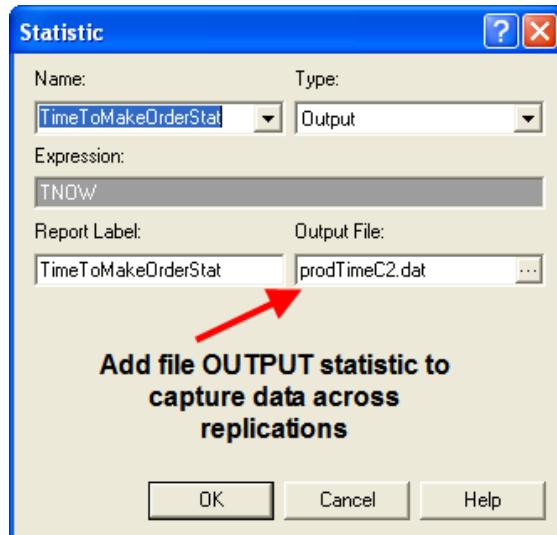


Figure 4.87.: Capturing total production time across replication results to a file

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 4.88 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 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 4.89, 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.

Based on these results, LOTR Makers Inc should consider sharing the rework worker between the work stations. If you check the other performance measures as per Figures 4.90 and 4.91, 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.

4. Modeling Systems with Processes and Basic Entity Flow

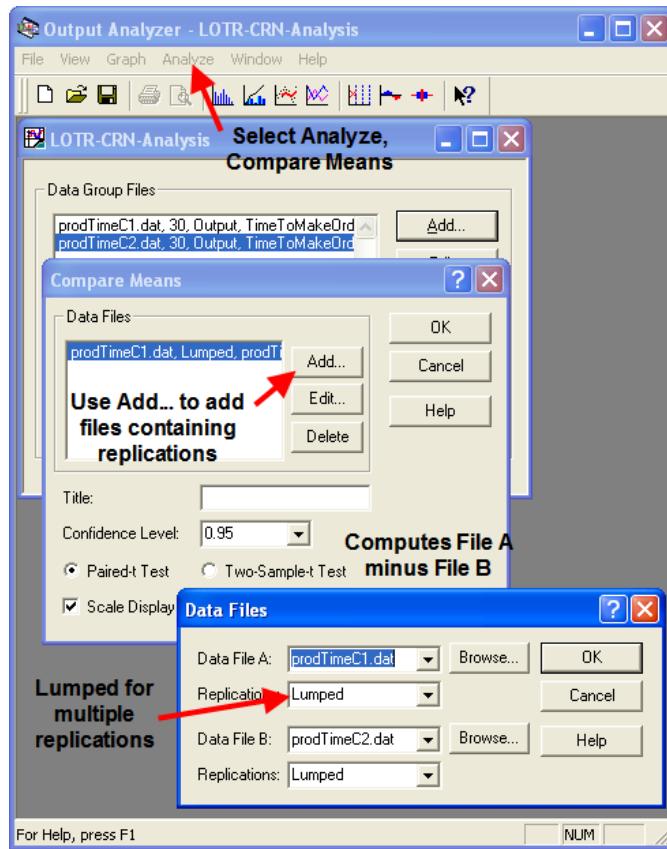


Figure 4.88.: Setup paired difference analysis in Output Analyzer

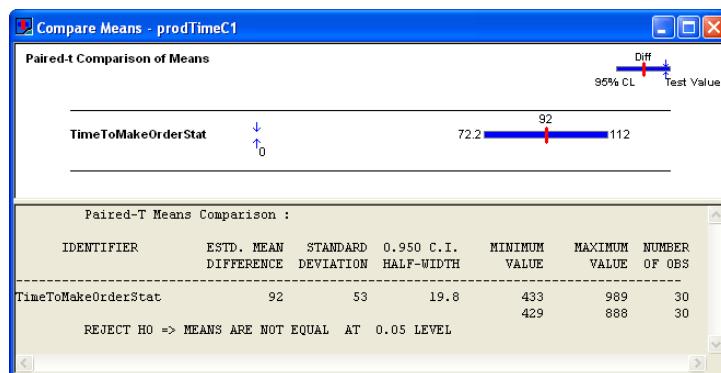


Figure 4.89.: Results for the paired difference analysis

4.6. Comparing Two Alternative Configurations for the LOTR Makers

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 4.90.: 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 4.91.: Configuration 2 resource utilization

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 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.

4.6.1.1. Implementing Independent Sampling

This section outlines how to perform the independent sampling case within a model. The completed model files are available as *LOTRConfig1IND.doe* and *LOTRConfig2IND.doe* in the folder called *ComparingTwoSystemsIND* in the book support files for this chapter.

The basic approach to implementing 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 *LOTRConfig1IND.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)`
- Preparation and travel time distribution: `EXPO(12,vStream)`

4. Modeling Systems with Processes and Basic Entity Flow

- 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 *LOTRConfig2IND.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 4.92 presents the results from the analysis.

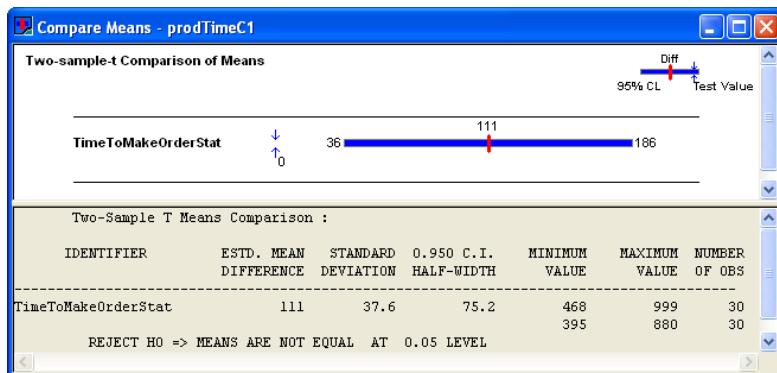


Figure 4.92.: Independent two sample analysis from Output Analyzer

The results indicate that configuration 2 has a smaller production time. Notice that the confidence interval for the independent analysis is wider than in the case of common random numbers. This indicates that there is more variability in the independent analysis than in the analysis that used 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 re-execute 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 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 analysis 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 pair-wise 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.

The last few sections discussed systems that involved queues. In fact, almost all of the models that have been presented within the previous chapters involved queueing lines of some kind. In the next section, we will look at ways to leverage the STATION and ROUTE modules to model more complex networks of queues. This will allow the scaling up of models and facilitate many common modeling situations, found especially in manufacturing environments.

4.7. Modeling Systems with Routing Sequences

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, milling, etc.). 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 4.93 illustrates the concept of a network of stations 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 have already been examined that can be considered queuing networks (e.g. the Tie-Dye T-Shirts and the LOTR's Ring Making examples). The purpose of this section will be to introduce some constructs that facilitate the simulation of queuing networks. Since a

4. Modeling Systems with Processes and Basic Entity Flow

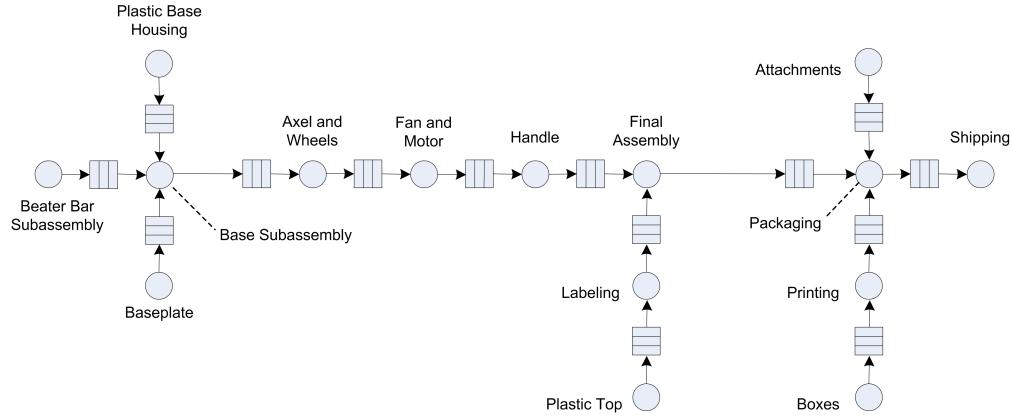


Figure 4.93.: Vacuum cleaner manufacturing system as a network of stations

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.

4.7.1. Computer Test and Repair Shop Example

Consider a test and repair shop for computer parts (e.g. circuit boards, hard drives, etc.) The system consists of an initial diagnostic station through which all newly arriving 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 2 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, 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 sequences in Table 4.4 constituted the vast majority of test plans for the parts.

Table 4.4.: Test plan sequences

Test Plan	% 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 test plan's 1 and 3 both 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 lognormal distribution with mean, μ , and standard deviation, σ in minutes. Table 4.5 presents the mean and standard deviation for each of the testing time distributions for each station in the test plan.

Table 4.5.: Testing and repair distributions

Test Plan	Testing Time Parameters	Repair Time Parameters
-----------	-------------------------	------------------------

Test Plan	Testing Time Parameters	Repair Time Parameters
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 3 workers that 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 to 4 minutes uniformly distributed. Figure 4.94 illustrates the arrangement of the stations and the flow of the parts following Plan 2 in the test and repair shop.

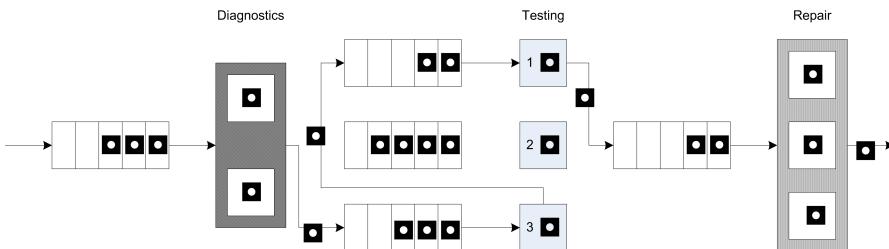


Figure 4.94.: Overview of the test and repair shop

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 2 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.

4.7.2. Conceptualizing the Model

Before implementing the model, you should prepare by conceptualizing the process flow. Figure 4.95 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

4. Modeling Systems with Processes and Basic Entity Flow

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 1 of 3 repair workers is seized for the repair activity. After the repair activity, the part leaves the system.

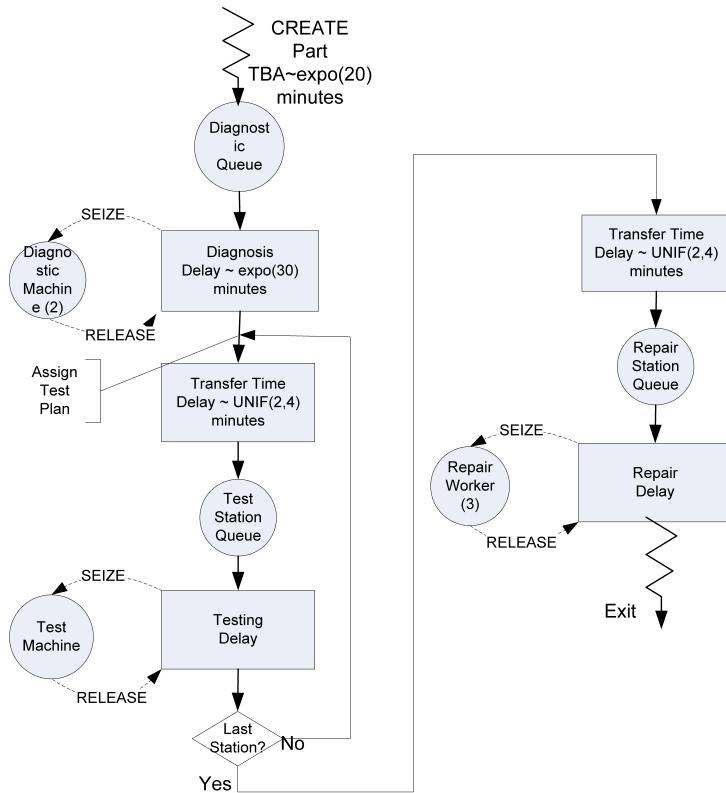


Figure 4.95.: Activity diagram for test and repair system

The following pseudo-code represent the main concepts for modeling the test and repair system. This is a straight forward 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 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). Secondly, 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.

```
CREATE part  
BEGIN PROCESS Diagnostics  
    SEIZE 1 diagnostic machine  
    DELAY for diagnostic time
```

```

RELEASE diagnostic machine
END PROCESS
BEGIN ASSIGN
    determine test plan type
    determine SEQUENCE for test plan type
END ASSIGN
ROUTE for transfer time by SEQUENCE to STATION Test

STATION Test
BEGIN PROCESS Testing
    SEIZE appropriate test machine
    DELAY for testing time
    RELEASE test machine
END PROCESS
DECIDE IF not at last station
    ROUTE for transfer time by SEQUENCE to STATION Test
ELSE
    ROUTE for transfer time by SEQUENCE to STATION Repair
END DECIDE

STATION Repair
BEGIN PROCESS Repair Process
    SEIZE repair worker from repair worker set
    DELAY for repair time
    RELEASE repair worker
END PROCESS
RECORD statistics
DISPOSE

```

The pseudo-code introduces some new concepts involving sequences, routes, and stations. The next section introduces these very useful constructs.

4.7.3. STATION, ROUTE, and SEQUENCE Modules

Entities typically represent the things that are flowing through the system. *Entity transfer* refers to the various ways by which entities can move between modules. As we have previously seen, 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.

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

4. Modeling Systems with Processes and Basic Entity Flow

stations, a SEQUENCE permits the listing of a set of assignments (e.g. ASSIGN modules) to be executed when the transfer occurs.

Unlike our previous use of the STATION and ROUTE module, we may need to know where (at which station) the entities are in order to provide appropriate processing logic. 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 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 5 different resources (`DiagnosticMachine`, `TestMachine1`, `TestMachine2`, `TestMachine3`, `RepairWorkers`) with capacity (2, 1, 1, 1, 3) respectively.
2. Define 2 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 3 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 4.96. First, double click on the SEQUENCE module to add a new sequence (row). The first sequence is called `TestPlan1Seq`. Then, add job steps to the sequence by clicking on the Steps button. Figure 4.96 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.

5. 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 4.97. 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 flow chart modules. The completed model can be found in the book support files associated with chapter called, *RepairShop.doe*. Figure 4.98 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

4. Modeling Systems with Processes and Basic Entity Flow

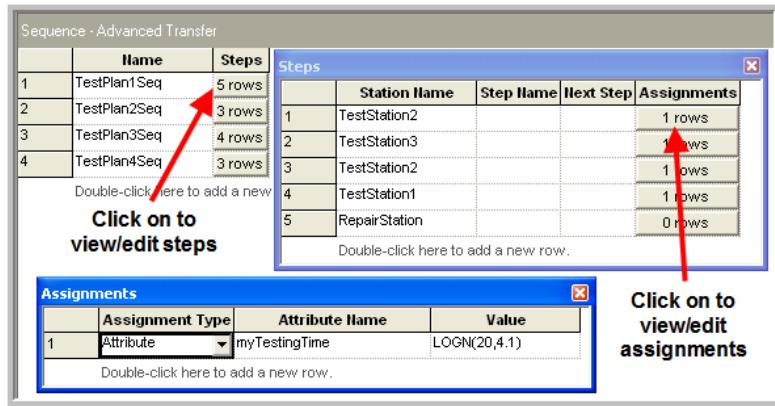


Figure 4.96.: Defining sequences, job steps, and assignments

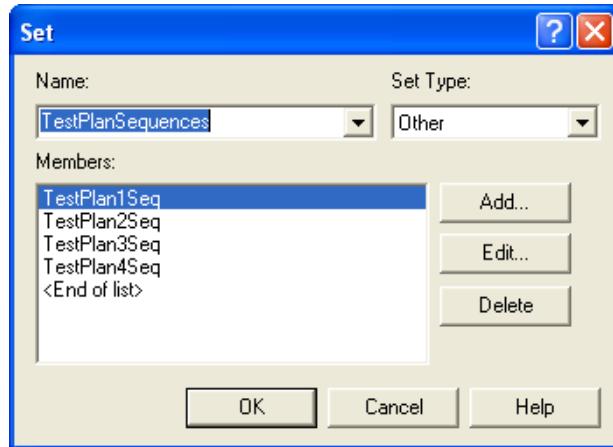


Figure 4.97.: Defining an advanced set to hold the sequences

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 4.99.

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 4.100 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 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 4.101 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 Se-*

4.7. Modeling Systems with Routing Sequences

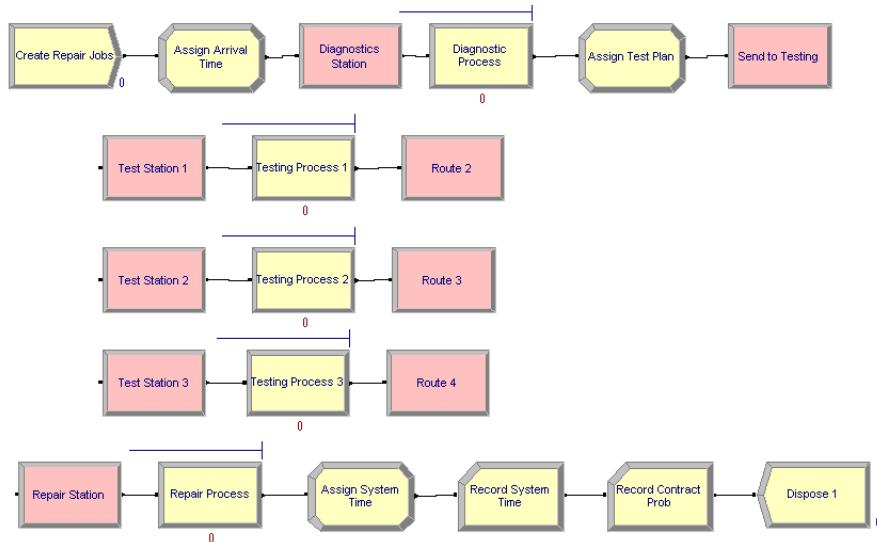


Figure 4.98.: Overview of the test and repair model

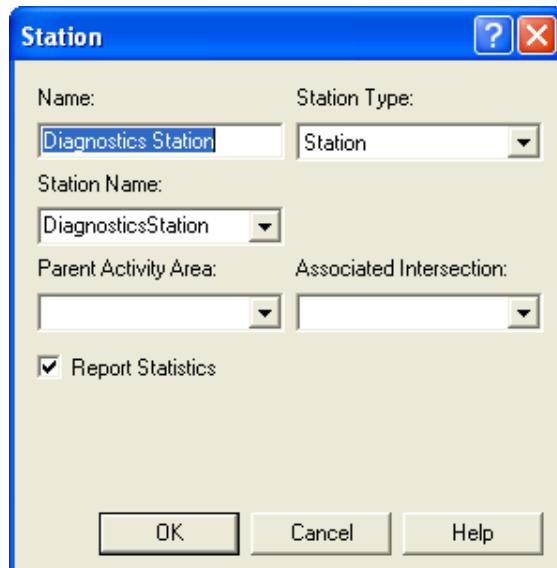


Figure 4.99.: Modeling the diagnostics station with a STATION module

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 4.100.: Assigning the test plans

4. Modeling Systems with Processes and Basic Entity Flow

quence 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 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.

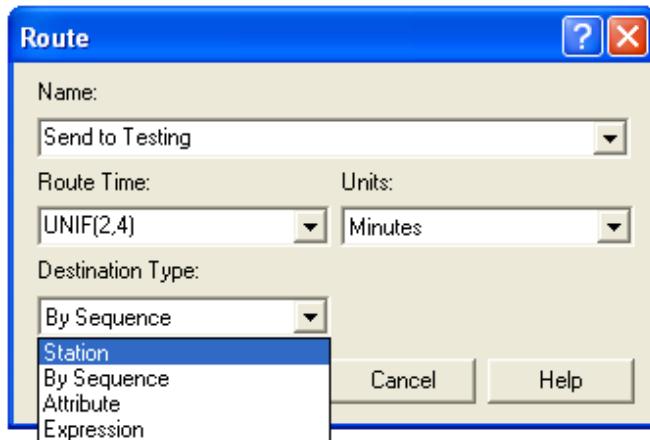


Figure 4.101.: Selecting the By Sequence option within the ROUTE module

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 (seize, delay, release), and ROUTE). The PROCESS module (Figure 4.102) 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.

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 4.103.

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 4.104.

4.7.4. Running the Test and Repair Model

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

4.7. Modeling Systems with Routing Sequences

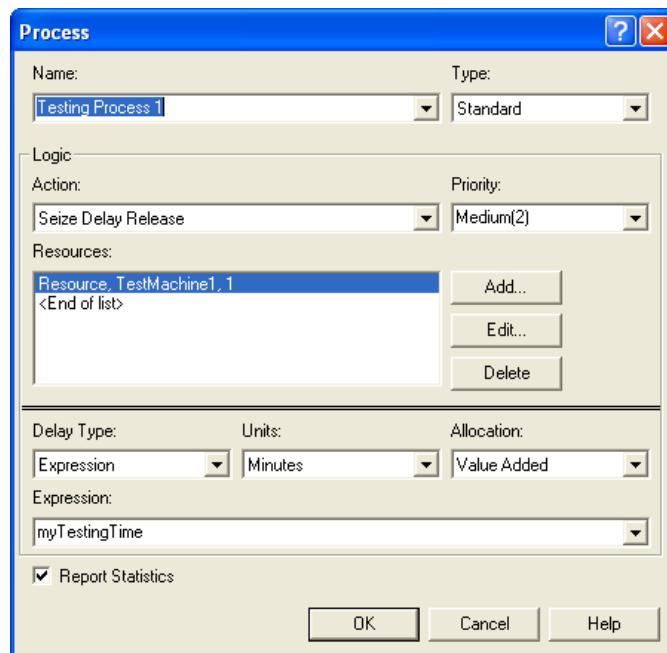


Figure 4.102.: PROCESS module for the testing process

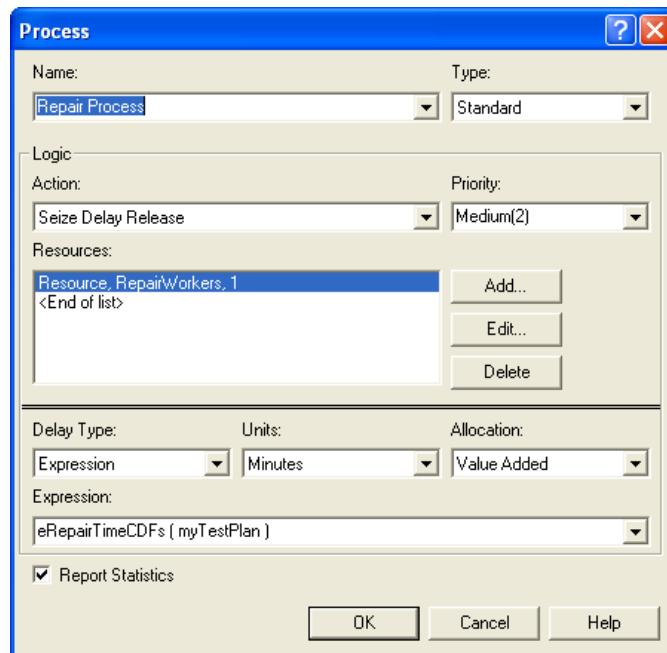


Figure 4.103.: PROCESS module for the repair process

4. Modeling Systems with Processes and Basic Entity Flow

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 4.104.: RECORD modules for test and repair model

working day, it is as if there are 249,600 minutes or 4,160 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 versus 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's assume that 10 replications of 4,160 hours will be sufficient as illustrated in Figure 4.105.

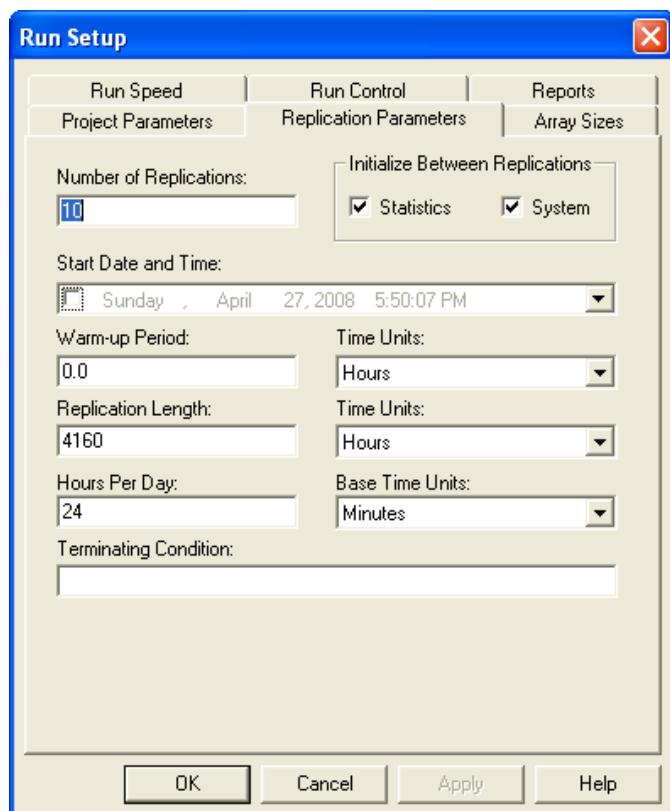


Figure 4.105.: Run setup specification for test and repair shop model

As shown in Figure 4.106, 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.

User Specified		
Tally		
Expression	Average	Half Width
ProbLTContractLimit	0.8251	0.02
SystemTimeStat	350.93	11.00

Figure 4.106.: User defined statistics for current contract

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, conveyors, etc.) 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 7.

4.8. Summary

In this chapter, you have learned a great deal about modeling processes within discrete-event dynamic systems using simulation. All of the modules found in Arena's Basic Process panel, except for the SCHEDULE module have been discussed. The modules covered included:

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.

4. Modeling Systems with Processes and Basic Entity Flow

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 that can be indexed by the location in the list.

In addition to Arena's Basic Process panel, the following constructs from the Advanced Process Panel 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.

We also saw how to access the basic programming blocks available on the BLOCKS panel via the WHILE-ENDWHILE blocks.

WHILE-ENDWHILE From the Blocks panel, these modules allow for iterative looping.

Finally, we illustrated the following modules from the Advanced Transfer Panel.

STATION module Allows the marking in the model for a location to which entities can be directed for processing.

SEQUENCE module Allows for pre-specified 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.

With all these modules, you can already model very complex systems.

In addition to systems modeling, we learned that when developing a simulation model, we are really developing software programs. To develop a program in Arena, you used data modules to define the elements to be used in the model and flow chart modules to specify the logical flow of the model. The flow chart style that 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 flow chart 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 computer. 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 beginning 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 LABELS, STATIONS, and sub-models to organize your code into manageable and logically consistent pieces. You should treat simulation model development 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 5 will return to some statistical concepts as we learn how to simulate infinite horizon systems. will concentrate on building models that incorporate randomness and how facilitates modeling random processes. Then, we will explore some of the more advanced modules within Arena. Along the way, more of the modules and concepts needed to build more realistic simulation models will be presented.

4.9. Exercises

Exercise 4.1. 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.

Exercise 4.2. 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 4 will want T-Shirts, a 30% chance that a family of 3 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.

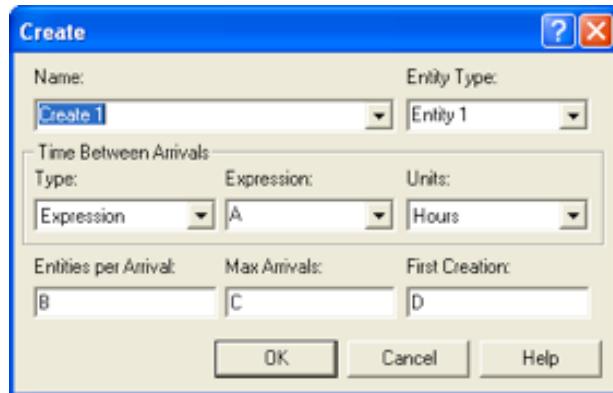


Figure 4.107.: CREATE Module

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: _____
-

Exercise 4.3. 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 and estimate the probability that a customer will balk because the line is too long. Run your model for 1 year, with 20 replications. *Hint* Use the NQ0 function.

Exercise 4.4. Samples of 20 parts from a metal grinding process are selected every hour. Typically 2% of the parts need 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 simulate 30 hours of the process, i.e. 30 samples of size 20, and estimate the chance that X exceeds its expected value by more than 1 standard deviation.

Exercise 4.5. Samples of 20 parts from a metal grinding process are selected every hour. Typically 2% of the parts need 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 1 standard deviations. Each time X exceeds its mean by more than 1 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 1 worker at its repair process. Once both of the subcomponents have been repaired, they are joined back together to form the original part. The joining process takes 5 minutes with 1 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.

Exercise 4.6. 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 the average, 82 percent of the sets pass inspection. The remaining 18% are 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 a 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 TV is adjusted.

Exercise 4.7. A simple manufacturing system is staffed by 3 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 3 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 model for this situation. Report the average system time for the parts based on 20 replications of 4800 minutes.

Exercise 4.8. The Hog BBQ Joint is interested in understanding the flow of customers for dinner (5 pm to 9 pm). Customers arrive in parties of 2, 3, 4, or 5 with probabilities 0.4, 0.3, 0.2, 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 2 people. For parties, with more than 2 customers, the tables are moved together. Each arriving group gets in line to be seated. If there are already 6 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 a 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		

Exercise 4.9. 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. Re-run the model for 30 eight hour days with the arrival rate increased by 20, 40, 60, and 80 percent. Will the system have trouble meeting the demand? Use the statistics to justify your conclusions.

Exercise 4.10. 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 percent 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. Re-run the model for 30, 8-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?

Exercise 4.11. 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 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 (plus or minus 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 average 45 seconds plus or minus 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 2nd window, then the order must wait for the customer. After both the order and

4. Modeling Systems with Processes and Basic Entity Flow

the customer are at the 2nd window, the 2nd 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.

Exercise 4.12. 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, electronic equipment, etc.) 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% are 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 to 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 that 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 to 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.

Exercise 4.13. 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 place 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 minutes plus or minus 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 1 operator. There could be a quality problem with the pole or with 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 two, 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.

Exercise 4.14. 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

4. Modeling Systems with Processes and Basic Entity Flow

arrival, patients are assigned a rating of 1 to 5, depending on the severity of their ailments. Patients in Category 1 are the most severe, and they 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
Percent	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:

- Average number of patients waiting for registration
- Utilization of beds
- System time of each type of patient and overall (across patient types)

Using a run length of 30 days, develop a 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.

Exercise 4.15. Customers enter a fast-food restaurant according to an exponential inter-arrival 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, 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		

Exercise 4.16. Create a model 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 $\mu_1 = 2$, $\sigma_1^2 = 0.64$ and $\mu_2 = 2.2$, $\sigma_2^2 = 0.64$. Assume that you don't know the true means and variances. Use the method of independent samples to test whether $\mu_2 > \mu_1$.

Exercise 4.17. Create a model 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 $\mu_1 = 2$, $\sigma_1^2 = 0.64$, and $\mu_2 = 2.2$, $\sigma_2^2 = 0.36$. Assume that you don't know the true means and variances. Use the method of independent samples to test whether $\mu_2 > \mu_1$.

Exercise 4.18. Create a model to simulate observations from a $N(\mu, \sigma^2)$ random variable. Use your simulation to generate two independent samples of size $n_1 = 30$ and $n_2 = 30$ from normal distributions having $\mu_1 = 2$, $\sigma_1^2 = 0.64$ and $\mu_2 = 2.2$, $\sigma_2^2 = 0.36$. Assume that you don't know the true means and variances. Use the paired-t method to test whether $\mu_2 > \mu_1$.

Exercise 4.19. Create a model to simulate observations from a $N(\mu, \sigma^2)$ random variable. Use your simulation to generate two dependent samples of size $n_1 = 30$ and $n_2 = 30$ from normal distributions having $\mu_1 = 2$, $\sigma_1^2 = 0.64$ and $\mu_2 = 2.2$, $\sigma_2^2 = 0.36$. Use the method of common random number. Assume that you don't know the true means and variances. Use the paired-t method to test whether $\mu_2 > \mu_1$.

Exercise 4.20. Jobs arrive in batches of ten items each. The inter-arrival time is EXPO(2) hours. The machine shop contains 2 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 5 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.

Exercise 4.21. 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 3 items per hour, while the inspector can inspect 8 items per hour each exponentially distributed. Approximately 10% of all items fail inspection and are 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 2 shifts per day for 5 days. Any jobs not completed by the end of 2nd shift are carried over to the 1st 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 +/- 3 minutes.

- The average number of jobs completed per week.
- a. Sketch an activity diagram for this situation.
 - b. Assume that there are 2 technicians at the repair station, 1 inspector at the inspection station, and 1 technician at the disassembly station. Develop a model for this situation.
 - c. Assume that there are 2 technicians at the repair station and 1 inspector at the inspection station. The disassembly station is also staffed by the 2 technicians that are assigned to the repair station. Develop a model for this situation.
-

Exercise 4.22. 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, 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 has 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 log-normally 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 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 using stream 6.

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 of our recommendation to within 2 minutes.

Exercise 4.23. 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 data set associated with Exercise B.13.

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 1 customer every 40 seconds. The number of copies requested by each customer is equally likely over the range of 10 and 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 queue 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 develop 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 a 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 spends in the system is longer than 4 minutes.

Exercise 4.24. 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 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 have 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 versus non-frequent 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		
Non-Frequent 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		

4. Modeling Systems with Processes and Basic Entity Flow

- 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 ± 1 minute with 95% confidence.
-

Exercise 4.25. 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	% 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)

Management is 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 2 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.

Exercise 4.26. Parts arrive at a 4 workstation system according to an exponential inter-arrival distribution with a mean of 10 minutes. The workstation A has 2 machines. The three workstations (B, C, D) each have a single machine. There are 3 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 30000 minutes and discuss the potential bottlenecks in the system.

5. Statistical Analysis for Infinite Horizon Simulation Models

LEARNING OBJECTIVES

- To understand the concept of steady state analysis and the implications of modeling an infinite horizon system.
- To be able to analyze infinite horizon simulations via the method of replication-deletion
- To be able to analyze infinite horizon simulations via the method of batch means
- To be able to perform sequential sampling for an infinite horizon model
- To be able to apply verification and validation methods

This chapter discusses how to plan and analyze infinite horizon simulations. When analyzing infinite horizon simulations, the primary difficulty is the nature of observations from within a replication. 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)

In Chapter 3, these requirements were met by performing independent replications of the simulation to generate a random sample. Simulating independent replications is natural when performing an analysis of a finite horizon model. In the case of a finite horizon model, there are very well determined initial and ending conditions. Initial conditions refer to the starting conditions for the model, i.e. whether or not the system starts "empty and idle". The challenge of performing an infinite horizon simulation is that there are, in essence, no initial and ending conditions. We often call infinite horizon simulation by the term steady state simulation. This is because, in this context, we want to measure the performance of the system after a long time (i.e. under steady state conditions). The effect of initial conditions on steady state performance measures will be discussed in this chapter.

In a direct sense, the outputs from within a replication do not satisfy the requirements of producing a random sample (IID observations); however, certain procedures can be imposed on

5. Statistical Analysis for Infinite Horizon Simulation Models

the manner in which the observations are gathered to ensure that these statistical assumptions are not grossly violated. The following section will first explain why within replication observations typically violate these assumptions and then we will explore some methods for mitigating the violations within the context of infinite horizon simulations.

The chapter ends with an illustration of methods to verify and validate a simulation model. The analysis will rely on concepts of queueing theory (see Appendix C) and the use of analytical formulas that apply to steady state queueing systems.

5.1. A Spreadsheet Example

To illustrate the challenges related to infinite horizon simulations, a simple spreadsheet simulation was developed for a M/M/1 queue as introduced in Chapter 2. The spreadsheet model is provided in the spreadsheet file *MM1-QueueingSimulation.xlsxm* 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 5.1.

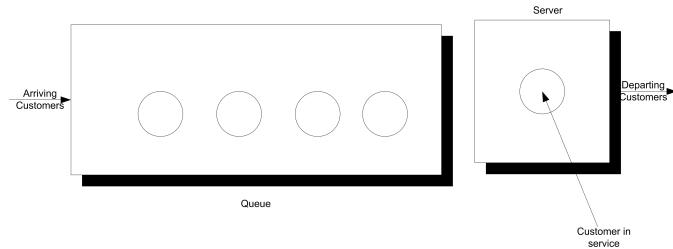


Figure 5.1.: Single server queueing system

For a single server queueing 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 inter-arrival times for each of the customers that visit the queue. Let $E[Y_i] = 1/\lambda$ be the mean of the inter-arrival times so that λ is the mean arrival rate. Let $E[X_i] = 1/\mu$ be the mean of the service times so that μ 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:

$$W_{i+1} = \max(0, W_i + X_i - Y_i)$$

The relationship says that the time that the $(i + 1)^{st}$ 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), less the time between arrivals of the i^{th} and $(i + 1)^{st}$ customers, Y_i . If $W_i + X_i - Y_i$

5.1. A Spreadsheet Example

is less than zero, then the $((i + 1)^{st}$ customer arrived after the i^{th} finished service, and thus the waiting time for the $(i + 1)^{st}$ customer is zero, because his service starts immediately.

Suppose that $X_i \sim \exp(E[X_i] = 0.7)$ and $Y_i \sim \exp(E[Y_i] = 1.0)$. This is a M/M/1 queue with $\lambda = 1$ and $\mu = 10/7$. Thus, using the equations given in Chapter 2 or Appendix C, yields:

$$\rho = 0.7$$

$$L_q = \frac{0.7 \times 0.7}{1 - 0.7} = 1.6\bar{3}$$

$$W_q = \frac{L_q}{\lambda} = 1.6\bar{3} \text{ minutes}$$

The spreadsheet model involves generating X_i and Y_i as well as implementing Lindley's equation within the rows of the spreadsheet. Figure 5.2 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 5.3.

A	B	C	D	E	F	G	H
20							
21	Customer Number	Waiting time W(n)	Service time X(n)	Interarrival Y(n)		Cumulative Sum	Cumulative Avg
22	n						
24	0	0	0.165049697	3.205865303	-3.040815606		
25	1	0	0.52814479	1.34872615	-0.82058136	0	0
26	2	0	0.338501251	2.146635912	-1.808134661	0	0
27	3	0	1.435505306	1.442529663	-0.007024357	0	0
28	4	0	0.436372252	0.184805878	0.251566375	0	0
29	5	0.251566375	0.600203813	0.576593493	0.275176695	0.251566375	0.050313275
30	6	0.275176695	1.499580523	3.497237695	-1.722480477	0.52674307	0.087790512
31	7	0	2.087126617	0.225136044	1.861990574	0.52674307	0.07524901
32	8	1.861990574	0.940140227	4.372192785	-1.570061985	2.388733644	0.298591705
33	9	0	0.55141614	0.710258739	-0.158842598	2.388733644	0.265414849
34	10	0	0.489774047	0.0205816	0.469192446	2.388733644	0.238873364

Figure 5.2.: Spreadsheet for Lindley's equation

Figure 5.3 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

$$\frac{1}{n} \sum_{i=1}^n W_i \text{ for } n = 1, 2, \dots$$

5. Statistical Analysis for Infinite Horizon Simulation Models

	A	B	C	D	E	F	G	H
20								
21	Customer Number	Waiting time	Service time	Interarrival time				
22	n	W(n)	X(n)	Y(n)				
23							Cumulative Sum	Cumulative Avg
24	0	0	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C24+D24-E24			
25	1	=MAX(0,F24)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C25+D25-E25	=C25		=G25/B25
26	2	=MAX(0,F25)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C26+D26-E26	=C26+G25		=G26/B26
27	3	=MAX(0,F26)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C27+D27-E27	=C27+G26		=G27/B27
28	4	=MAX(0,F27)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C28+D28-E28	=C28+G27		=G28/B28
29	5	=MAX(0,F28)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C29+D29-E29	=C29+G28		=G29/B29
30	6	=MAX(0,F29)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C30+D30-E30	=C30+G29		=G30/B30
31	7	=MAX(0,F30)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C31+D31-E31	=C31+G30		=G31/B31
32	8	=MAX(0,F31)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C32+D32-E32	=C32+G31		=G32/B32
33	9	=MAX(0,F32)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C33+D33-E33	=C33+G32		=G33/B33
34	10	=MAX(0,F33)	=-\$B\$4*LN(1-RAND())	=-\$B\$5*LN(1-RAND())	=C34+D34-E34	=C34+G33		=G34/B34

Figure 5.3.: Spreadsheet formulas for queueing simulation

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 customer waiting times.

Figure 5.4 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 5.4 indicates that the sample average is significantly lower than the true expected average. Figure 5.5 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 towards 1.2 minutes.

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 5.4.: Results across 1000 customers

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 5.6, it is readily apparent that the data has strong positive correlation using the methods discussed in Appendix B. The lag-1 correlation for this data is estimated to be about 0.9. Figure 5.7 clearly indicates the strong

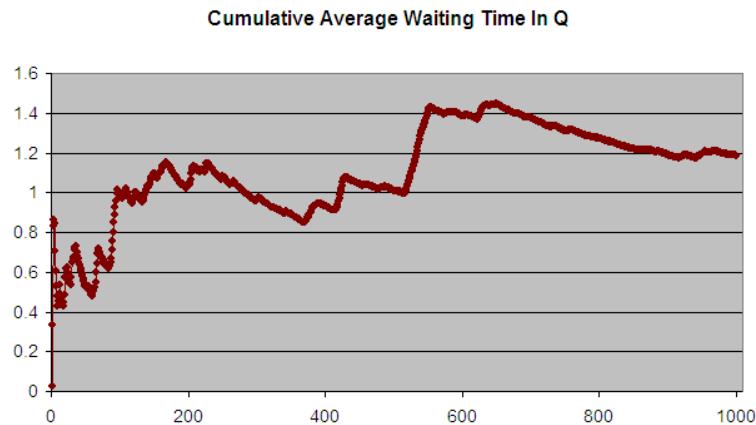


Figure 5.5.: Cumulative average waiting time of 1000 customers

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 a long time. If the previous customer had a short wait, then the next customer is likely to have a short wait. This makes sense with respect to how a queue operates.

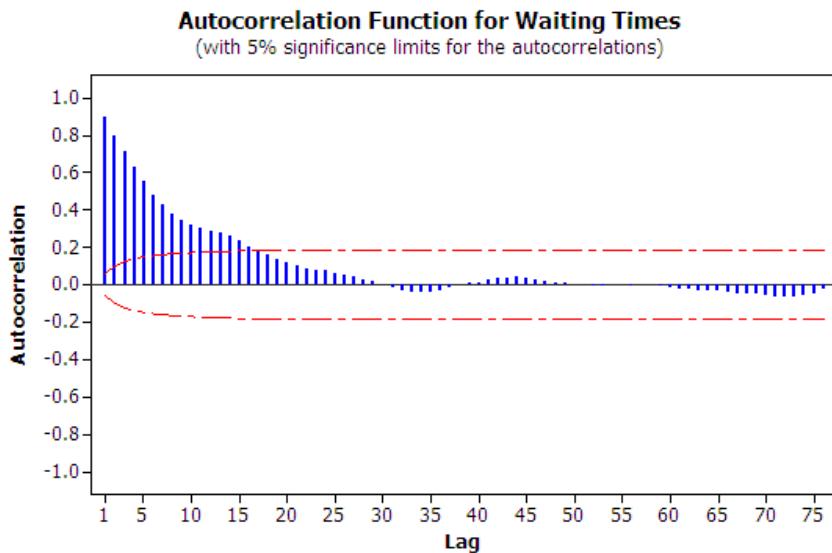


Figure 5.6.: Autocorrelation plot for waiting times

Strong positive correlation has serious implications when developing confidence intervals on the mean customer waiting time because the usual estimator for the sample variance:

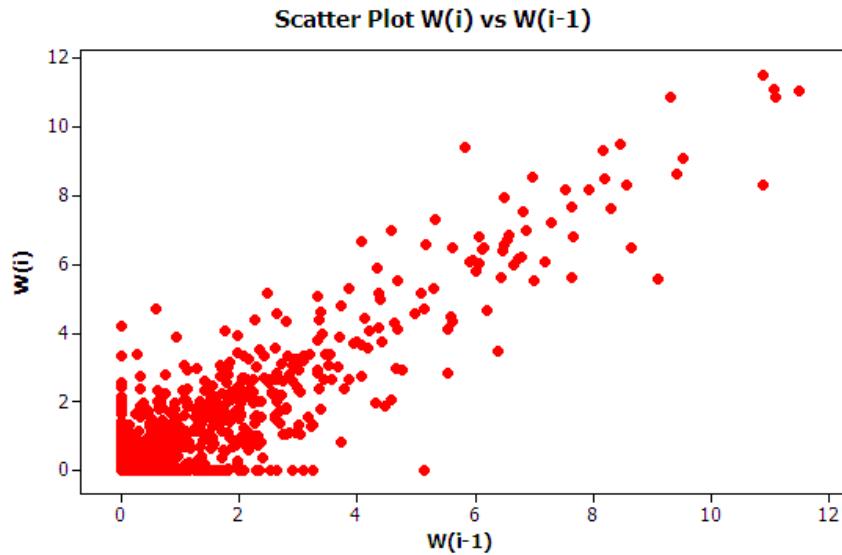


Figure 5.7.: Lag 1 dependence for waiting times

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

is a **biased** estimator for the true population variance when there is correlation in the observations. This issue will be re-examined 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 Appendix B, 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 was 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 than those faced by the 1000th customer. Thus, the distributions of their waiting times are likely to be different.

Recall the definition of covariance stationary from Appendix B. 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 $Var[X_i] = \sigma^2 > 0$, for $i=1,2,\dots, n$
- The lag-k autocorrelation, $\rho_k = cor(X_i, X_{i+k})$, is not a function of i , i.e. 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 $Var[X_i] \neq \sigma^2$ 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_{10000} will be similar? The argument, that the 9999th customer is on average likely to experience similar conditions as the 10000th customer, sure seems reasonable. Figure 5.8 shows 10 different replications of the cumulative average for a 10000 customer simulation. This was developed using the sheet labeled 10000-Customers.

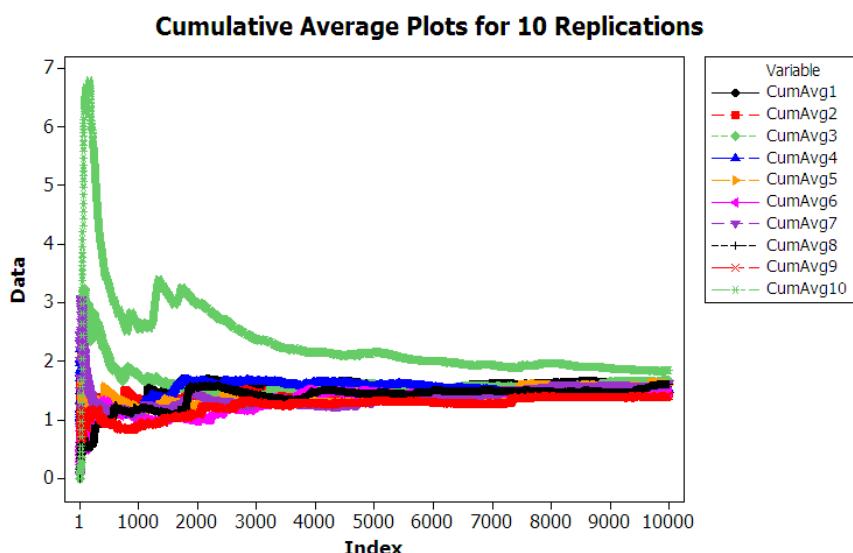


Figure 5.8.: Multiple sample paths of queueing simulation

From the figure, you can see that the cumulative average plots can vary significantly over the 10000 customers with the average tracking above the true expected value, below the true expected value, and possibly towards 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 10000 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* towards the true mean.

5. Statistical Analysis for Infinite Horizon Simulation Models

Determining the warm up time will be the basic way to mitigate the problem of non-identical 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 queue, 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 observations generated for this example. The problems associated with observations from within a replication need to be addressed in order to properly analyze infinite horizon simulation experiments.

5.2. Statistical Analysis Techniques for Warmup Detection

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 fashion 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.

5.2.1. 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, θ , 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 parameter of interest, θ , 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, θ , and a particular observation, X_i , is called error, $\epsilon_i = X_i - \theta$. 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 5.1 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 $W_q = 1.63\bar{3}$. 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, i.e. on average it is less than the target value.

Table 5.1.: 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
	$\bar{W} = 0.6797$	$\bar{B} = -0.9536$

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.

5. Statistical Analysis for Infinite Horizon Simulation Models

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 $X_i; i = d + 1, \dots$.

For example, when estimating the steady state mean waiting time for each replication r the estimator would be:

$$\bar{W}_r = \frac{1}{n-d} \sum_{i=d+1}^n W_{ir}$$

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:

$$\bar{Y}_r = \frac{1}{T_e - T_w} \int_{T_w}^{T_e} Y_r(t) dt$$

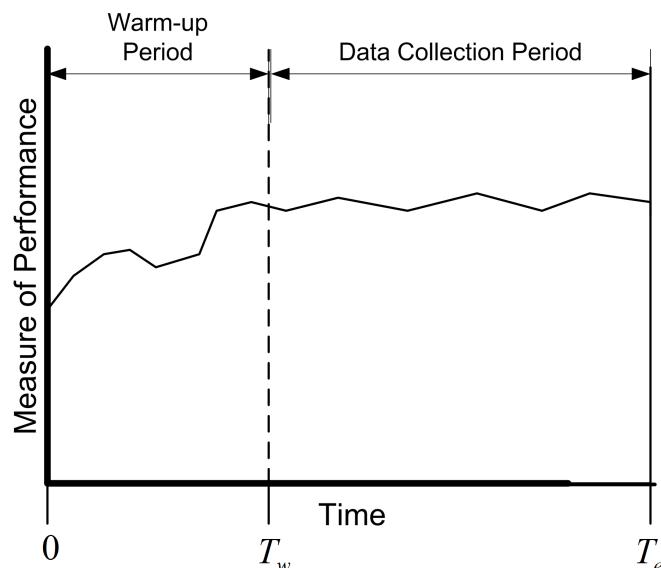


Figure 5.9.: The concept of the warm up period

Figure 5.9 show 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** to 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 at $T = 0$. For example, it is certainly possible that at time T_w for a particular replication that the system will be 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 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, 1983) called the Welch Plot.

5.2.2. Using a Welch Plot to Detect the Warmup Period

The basic idea behind Welch's graphical procedure is simple:

- Make R replications. Typically, $R \geq 5$ is recommended.

5. Statistical Analysis for Infinite Horizon Simulation Models

- 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 $r = 1, 2, \dots, n$.

$$\bar{Y}_{\cdot j} = \frac{1}{n} \sum_{r=1}^n Y_{rj}$$

- Plot, $\bar{Y}_{\cdot j}$ for each $j = 1, 2, \dots, m$
- Apply smoothing techniques to $\bar{Y}_{\cdot j}$ for $j = 1, 2, \dots, m$
- Visually assess where the plots start to converge

Let's 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 5.10, cell B2 represents the average across the 10 replications for the 1st customer. Column D represents the cumulative average associated with column B.

	A	B	C	D	E	F	G	H	I
1	Average								
2	1	0.548433	0.548433	0.548432719	0	0.337306	0	1.79165	0.716249
3	2	0.359163	0.907596	0.453798017	0.213726	0.427596	0	1.892545	0
4	3	0.793544	1.70114	0.567046583	0.428974	1.498441	0	2.921952	1.373886
5	4	0.679799	2.380939	0.595234777	0.141178	2.265391	0	1.559509	1.523948
6	5	0.772271	3.15321	0.630642059	0.255458	0.839788	0	1.930919	1.569935
7	6	0.598509	3.75172	0.625286585	0.022953	0.909185	0	1.677104	1.460226
8	7	0.760241	4.51196	0.644565755	0	0.917162	0	0.152849	1.979963
9	8	0.879366	5.391326	0.673915781	0	0.350159	0	0	3.596641
10	9	1.169247	6.560573	0.728952544	0.161749	0.457508	0	0	2.767113
11	10	0.897993	7.458566	0.745856589	0.361444	1.45561	0	0	1.213456

Figure 5.10.: Computing the averages for the Welch plot

Figure 5.11 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 the data. From the plot, you can infer that after about customer 3000 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 5.2.

5.2. Statistical Analysis Techniques for Warmup Detection

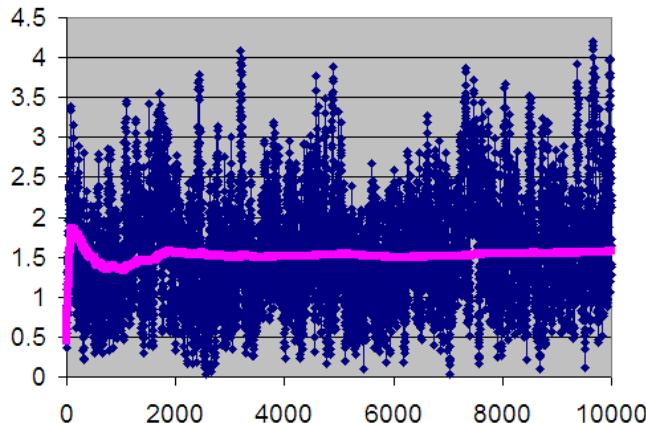


Figure 5.11.: Welch plot with superimposed cumulative average Line

Table 5.2.: Replication-Deletion results, $d = 3000$

r	$\bar{W}_r(d = 0)$	$\bar{W}_r(d = 3000)$	$B_r(d = 0)$	$B_r(d = 3000)$
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
	$\bar{W} = 1.573617$	$\bar{W} = 1.603595$	$\bar{B} = -0.05972$	$\bar{B} = -0.02974$
	$s = 0.1248$	$s = 0.1286$	$s = 0.1248$	$s = 0.1286$
95% LL	1.4843	1.5116	-0.149023	-0.121704
95% UL	1.6629	1.6959	-0.029590	0.062228

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.³¹⁹

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 towards 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 of dealing with a slow converging performance measure.

5. Statistical Analysis for Infinite Horizon Simulation Models

isfied 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 and OptQuest will be discussed in Chapter 8. The next section illustrates how to perform the method of replication-deletion on this simple M/M/1 model.

5.3. 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 for post processing
- Setup and run the multiple replications
- Make Welch plots based on the saved data
- Interpret the results

The file *MM1-ReplicationDeletionCSV.doe* contains the M/M/1 model for with the time between arrivals $\exp(E[Y_i]) = 1.0$ and the service times $\exp(E[Y_i]) = 0.7$ set in the CREATE and DELAY modules respectively. Figure 5.12 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 queueing time.

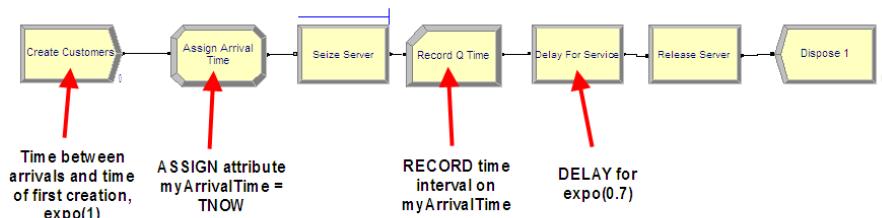


Figure 5.12.: MM1 Arena model

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 5.13 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 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 Arena's Output Analyzer. However, we have set the extension to (.csv) because we will post process the files using other programs.

	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

Figure 5.13.: STATISTIC Module with CSV extension added



Changing the extension of the output file is **not** sufficient to change the actual data type of the file that is written. We have set the extension in the STATISTIC module simply to ensure that the file name has the csv extension when saved to a file. Many programs will automatically recognize the CSV extension. In a moment, we will use the Run Setup > Run Control Advanced Settings (Figure 5.15) tab to ensure that the files are written out as text.

Figure 5.13 also 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 must be given that represents the value of the quantity to be observed through time. In this case, the expression builder has been used to select the current number of entities in the Seize 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 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 queueing simulations, the work by (Whitt, 1989) offers some ideas on specifying the run length, but these results are difficult to translate to general simulation models.

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) offer the rule of thumb that the run length should be at least 10 times the amount of data deleted. That is, $n \geq 10d$, where d is the number of observations deleted from the series. In terms of time, $T_e \geq 10T_w$, where T_w is the length of the warm up period in the time units of the simulation. 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 Welch plots etc.).

5. Statistical Analysis for Infinite Horizon Simulation Models

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 average) will it take for the simulation to generate d observations. In this model, the mean number of arrivals is 1 customer per minute. Thus, the initial T_w is

$$3000 \text{ customers} \times \frac{\text{minute}}{1 \text{ customer}} = 3000 \text{ minutes}$$

and therefore the initial T_e should be 30,000 minutes. Thus, in general, if we let η be the rate of occurrence of the observations, $\delta = 1/\eta$ be the time between occurrence, and d the desired number of observations to delete, we have that the length of the warm up period should be:

$$T_w = \frac{d}{\eta} = d \times \delta \quad (5.1)$$

Specify 30,000 minutes for the replication length and 10 replications on the Run > Setup > Replication Parameters as shown in Figure 5.14.

The replication length, T_e , is the time when the simulation will end. The Base Time Units can be specified for how the statistics on the default reports will be interpreted 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.

In order to process the recorded data from the simulation outside of Arena, you **must specify that the program produce a text file** rather than a *dat* file. To make this happen we need to indicated to Arena that output files should be written as text using the Run Setup > Run Control > Advanced dialog as shown in Figure 5.15. If this option is checked, all statistical data saved to files will be in a CSV format. Since we already ensured that the name of the file had the (.csv) extension this will facilitate Excel recognizing the file as a comma separate variable (CSV) file. Thus, when you double-click on the file, the file will be opened within programs that recognize the CSV format, such as Excel.

Running the simulation will generate two files *numInQ.csv* and *qtime.csv* within the current working directory for the model.

Figure 5.16 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 is written to the file. When multiple replications are executed, the number (-1) 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 fashion that is required for the analysis.

5.3. Performing the Method of Replication-Deletion

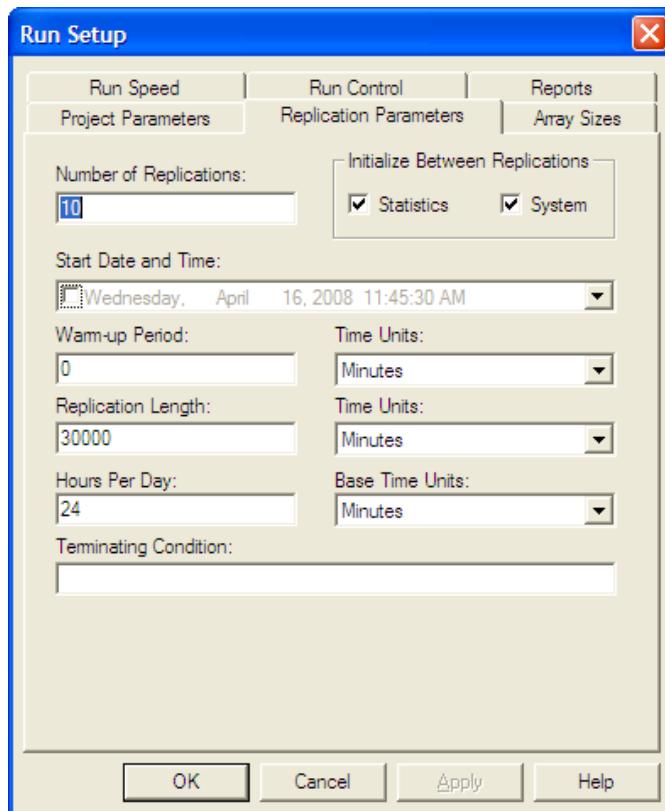


Figure 5.14.: Replication parameters for the warm up analysis

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. Thus, you may easily exceed the row limit of Excel by having too many replications and too many observations per replication. Regardless of whether or not the file can be *opened* in Excel, it still is a valid csv file and can be post processed.

To facilitate the making of Welch plots from Arena output files, I created a program that will read in the Arena text output file and allow the user to make a Welch plot as well as reformat the data to a more convenient CSV structure. The next section will overview the use of that software.

5.3.1. Looking for the Warm up Period in the Welch Plot Analyzer

As previously discussed, the Welch plot is a reasonable approach to assessing the warm up period. Unfortunately, the Arena does not automatically perform a Welch plot analysis. Before proceeding making Welch plots, there is still one more technical challenge related to the post-processing of time-persistent data that must be addressed.

5. Statistical Analysis for Infinite Horizon Simulation Models

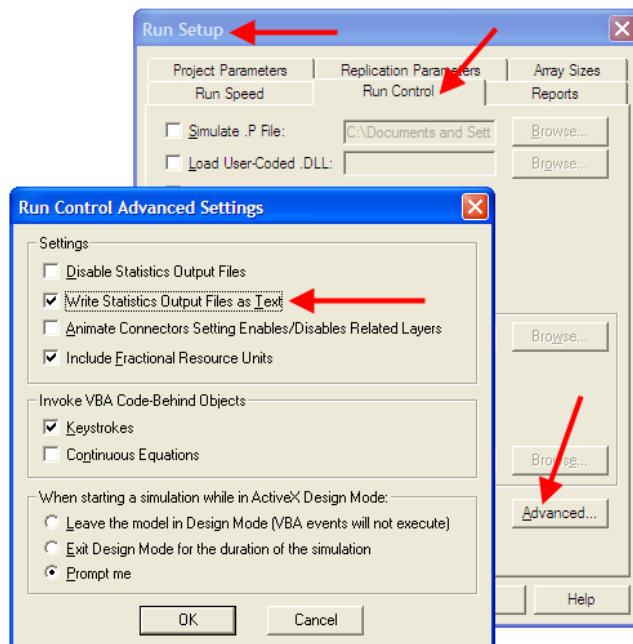


Figure 5.15.: Writing statistics output files as text.

	A	B	C	I
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		
9988	-1	0		End of replication indicator
9989	0.107442	0		

Figure 5.16.: Resulting CSV file as shown in Excel.

5.3. Performing the Method of Replication-Deletion

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 make a Welch plot, we need to discretize the observations into intervals of time that are equally spaced on the time axis.

Suppose that you divide T_e into k intervals of size Δt , so that $T_e = k \times \Delta t$. The time average over the j^{th} interval is given by:

$$\bar{Y}_{rj} = \frac{1}{\Delta t} \int_{(j-1)\Delta t}^{j\Delta t} Y_r(t) dt$$

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. Figure 5.17 illustrates the concept of discretizing time-persistent data.

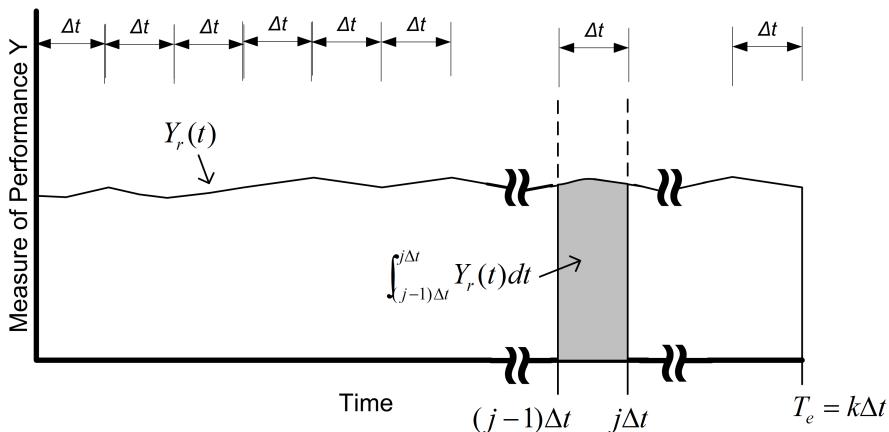


Figure 5.17.: Discretizing time persistent observations

The computation of the \bar{Y}_{rj} for time-persistent data can be achieved when processing the text file written by Arena.

Recall that the data for the Welch analysis is as follows, Y_{rj} is 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 $r =$

5. Statistical Analysis for Infinite Horizon Simulation Models

$1, 2, \dots, n$. The Welch averages are computed across the replications for each $j = 1, 2, \dots, m$, where $m = \min(m_r)$ for $r = 1, 2, \dots, n$.

$$\bar{Y}_{\cdot j} = \frac{1}{n} \sum_{r=1}^n Y_{rj}$$

Welch plot data has two components, the Welch averages ($\bar{Y}_{\cdot j}$ for $j = 1, 2, \dots, m$), and the cumulative average over the observations of the Welch averages:

$$\bar{\bar{Y}}_k = \frac{1}{k} \sum_{j=1}^k \bar{Y}_{\cdot j}$$

for $k = 1, 2, \dots, m$.

The Welch Plot Analyzer is a program that I created to facilitate making Welch plots from Arena data. You may find the Welch Plot Analyzer program files in the book support files. The Welch Plot Analyzer has an option for batching the data before making the Welch plot. Figure 5.18 presents the main dialog for the Welch Plot Analyzer. The user can perform four basic tasks:

1. Importing and batching observation or time-persistent data from an Arena generated CSV data file
2. Plotting an already processed Welch data file
3. Writing Welch plot data to a CSV file. The pairs $(\bar{Y}_{\cdot k}, \bar{\bar{Y}}_k)$ are written to the CSV file for each observation, $k = 1, 2, \dots, m$.
4. Writing all Welch data to a CSV file. The CSV file will contain each individual replication as a column, with the Welch average and cumulative average as the last two columns.

To begin using the Welch Plot Analyzer, use the top most *Browse* button to select the Arena CSV file that you want to process. After selecting the file, you should specify the type of statistic, either Tally or Time-Persistent. This choice will affect how the batching of the observations is performed.

If the data is tally based, then the batch size is interpreted as the number of observations within each batch. The total number observations, m , is divided into k batches of size b , where b is the batch size. It may be useful to batch the observations if m is very large. However, it is not critical that you specify a batch size other than the default of $b = 1$ for tally based data.

For time-persistent data, the batch size is interpreted as Δ as per Figure 5.17. It is useful to specify a Δ for discretizing the time-persistent data. The value of Δ will default to 1 time unit, which may not be adequate based on the rate of occurrence of the state changes for the time-persistent data. I recommend that Δ be set so that about 10 *observations* on average will fall within each interval. To get an idea for this you can simply open up the Arena generated CSV file and look at the first few rows of the data. By scanning the rows, you can quickly estimate about how much time there needs to be to get about 10 state changes.

5.3. Performing the Method of Replication-Deletion

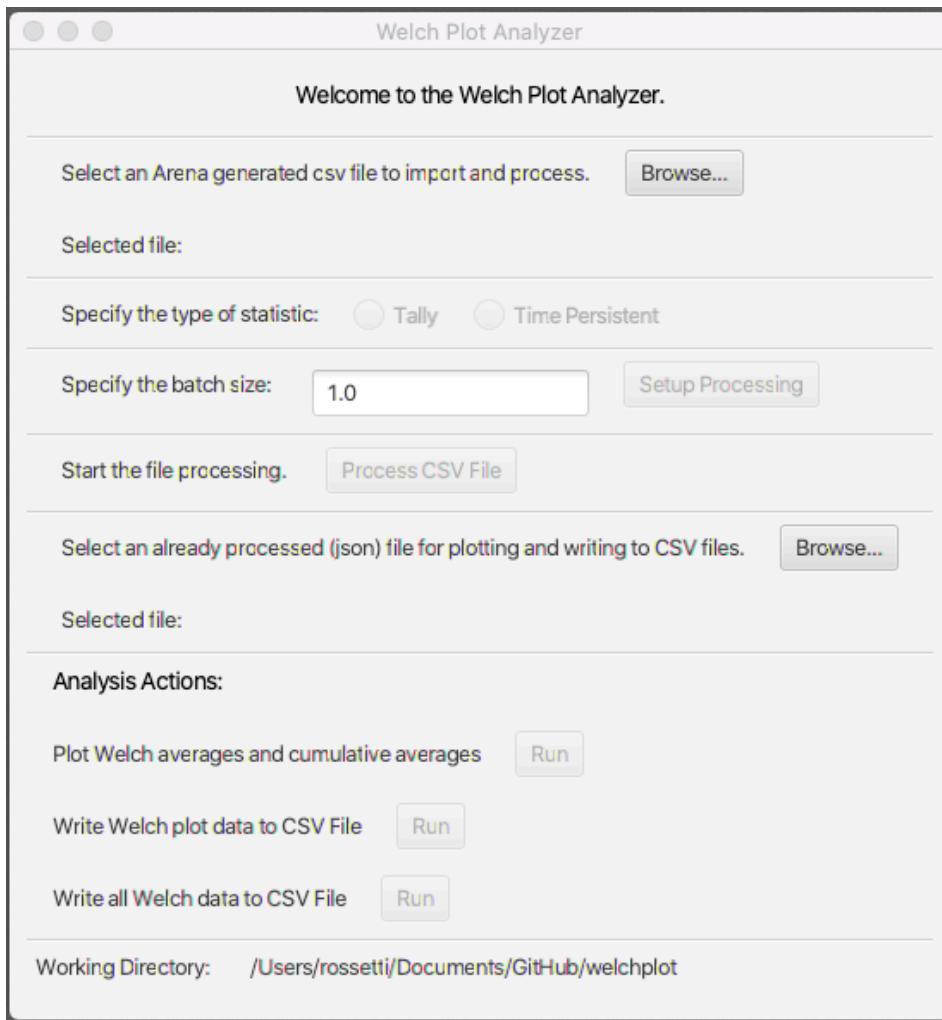


Figure 5.18.: Welch Plot Analyzer main dialog

Reviewing the data for the time-persistent number in queue data as shown in Figure 5.19 indicates that about 10 observations have occurred by time 5. Thus, it seems like having $\Delta = 5$ would be a reasonable setting.

After setting the batch size, the processing can be set up using the *Setup Processing* button. This will prepare the data for processing and check if the CSV file is a valid Arena text file. Then, the file can be processed by using the *Process CSV File* button. Depending on the size of the files, this action may take some time to complete. A dialog will appear when the operation is completed. The processing of the CSV file prepares the data for analysis actions by automatically populating the file for analysis. This file is a JSON file that holds meta-data (data about the data). A second file with a (.wdf) extension is produced that holds the underlying observations in a non-human readable form. The analysis action functionality of the Welch Plot Analyzer allows you to translate the data in the wdf file into a plot or into better organized csv files. Once the file has

	A	B	C
1	Project:	Unnamed Project	
2	User:		
3	Data item:	AvgNumInQ	
4	Run date:	5/25/21	
5	Options:	YDT	10
6			
7	Time	Observation	
8	0	0	
9	0.85154469	1	
10	0.88984738	0	
11	0.97643716	1	
12	1.76821142	2	
13	2.88858672	1	
14	3.68774303	2	
15	3.82884165	3	
16	4.17847559	2	
17	5.54558114	3	
18	5.66666951	4	
19	6.29760681	5	
20	6.35037849	4	

Figure 5.19.: Initial rows of number in queue data set.

been processed, the actions within the Analysis Actions section become available. By using the *Run* button to plot the Welch averages and cumulative averages we produce Figures 5.20 and 5.21. The Welch Plot Analyzer will automatically open up a web browser and display the plots. The plots are based on the [plotly¹](https://plotly.com/) open source java script plotting library. In addition, the plots are saved as *html* to a folder called *PlotDir*. You can open the *html* files at any time to review the plots.

Figures 5.20 and 5.21 provide the Welch plots for the time and queue and number in queue data. Notice that the title of the plot contains information that will convert each observation on the x-axis to units of time. For example, on Figure 5.21, we have that each observation is equivalent to 5 time units. That is because we discretized the data with $\Delta = 5$. Looking at the plot, it appears that the cumulative average converges after observation number 1000. Thus, to set the warm up period, we take $d = 1000$ and $\Delta = 5$ for $T_w = d \times \Delta = 1000 \times 5 = 5000$. In Figure 5.20, we see that there is a 1 to 1 correspondence between the observation number and time. This is because we get an arrival about every minute. Thus, looking at the plot, it seems that $d = 6000$

¹<https://plotly.com/>

5.3. Performing the Method of Replication-Deletion

is a good deletion point, which translated to $T_w = 6000$. We will set the warm up period of the simulation to the larger of the two T_w values (i.e. $T_w = 6000$)

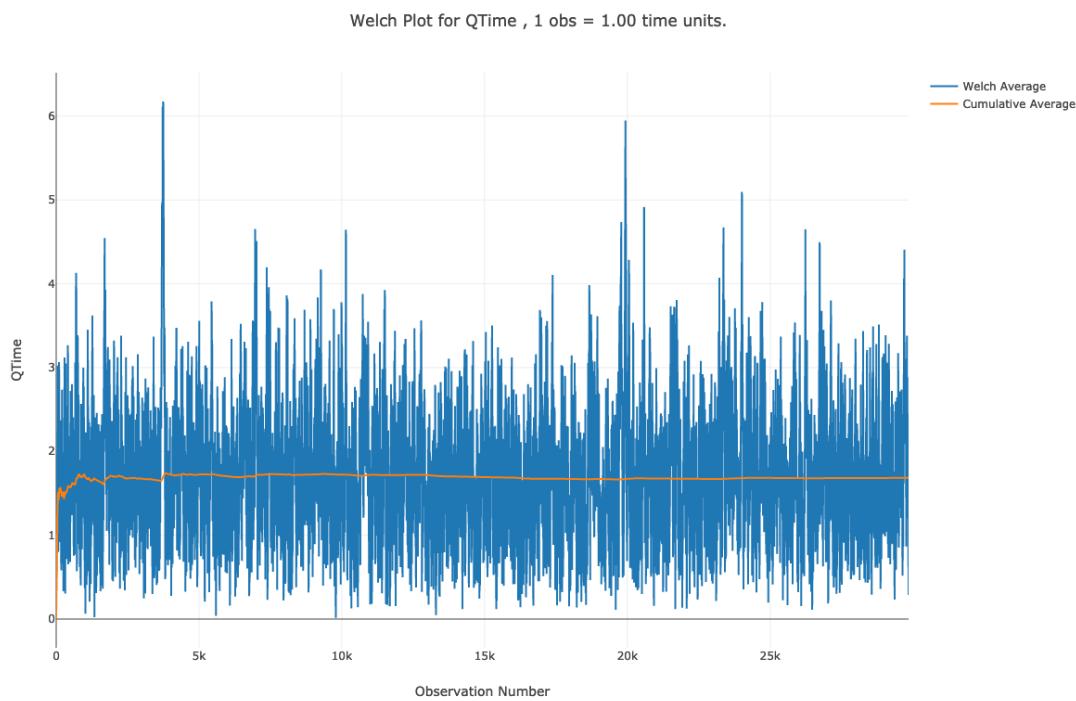


Figure 5.20.: Welch plot for time in queue data.

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. If re-running 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 re-run with the settings given in Figure 5.23 (30 replications, 6000 minute warm up period, 60,000 minute replication length). Since the true waiting time in the queue is 1.633, it is clear that the 95% confidence interval contains this value. Thus, the results shown in Figure 5.22 indicate that there does not appear to be any significant bias with these replication settings.

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.

5. Statistical Analysis for Infinite Horizon Simulation Models

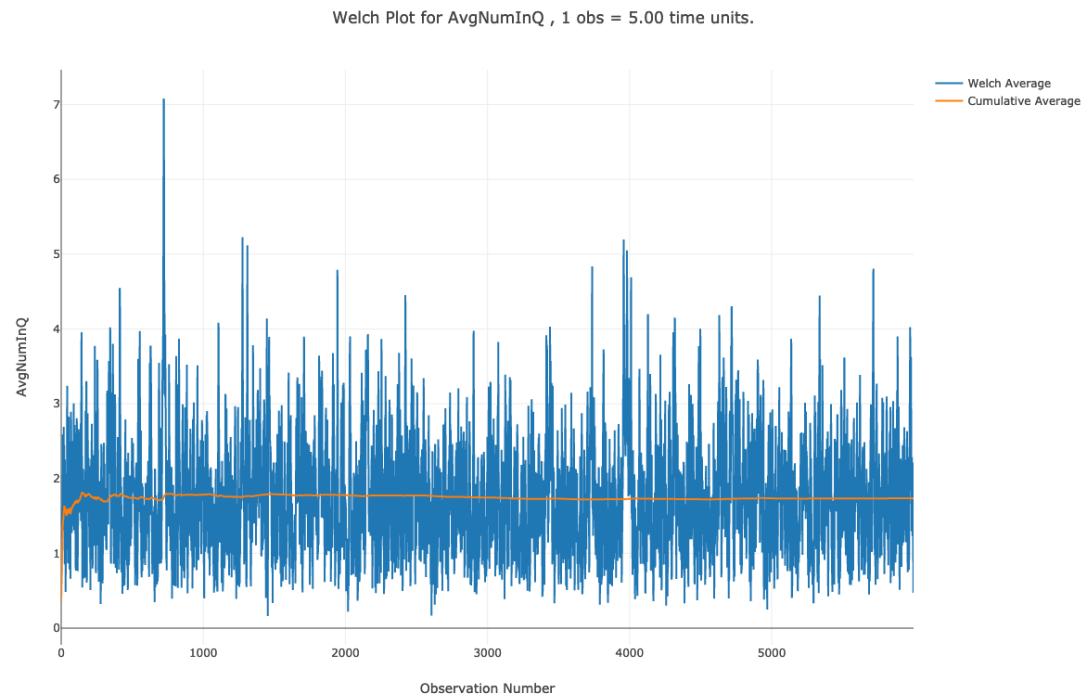


Figure 5.21.: Welch plot for number in queue data.

Waiting Time		Average	Half Width
Seize Server.Queue		1.6354	0.02
Other			
Number Waiting		Average	Half Width
Seize Server.Queue		1.6355	0.02

Figure 5.22.: Results based on 30 replications.

5.3. Performing the Method of Replication-Deletion

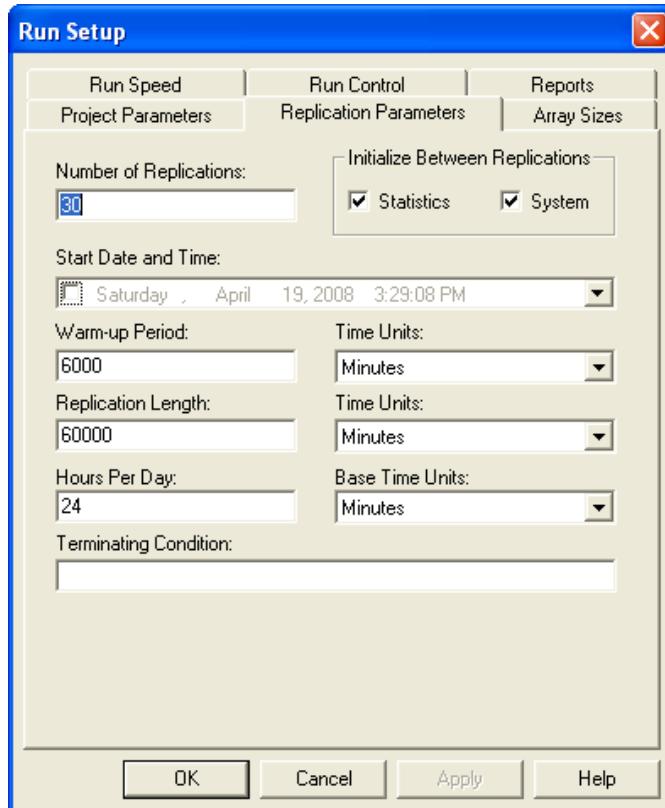


Figure 5.23.: Replication-Deletion settings for the example.

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: 1) increase the number of replications or 2) 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.

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 . The next section

5. Statistical Analysis for Infinite Horizon Simulation Models

discusses the statistical basis for the batch means method and addresses some of the practical issues of using it within Arena.

5.4. The Batch Means Method

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 5.24. In fact, this is essentially the same concept as previously discussed when processing the data using the Welch Plot Analyzer.

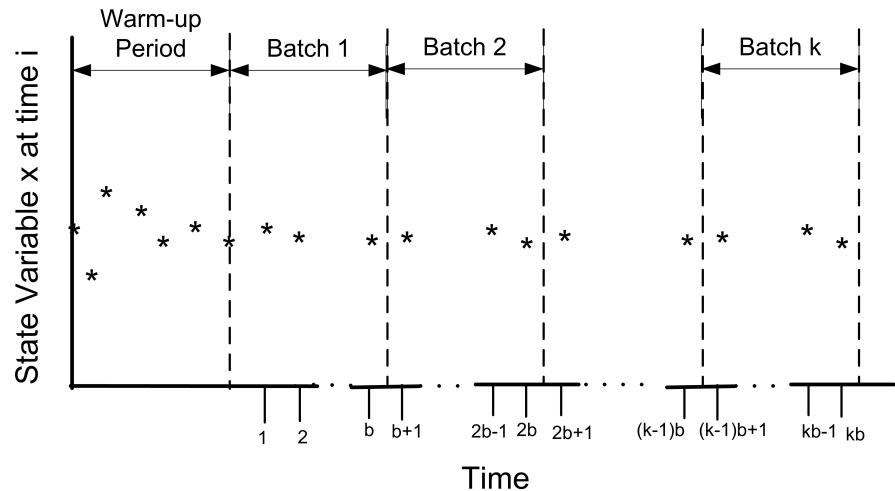


Figure 5.24.: Illustration of the batch means concept

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 5.3.1. 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 5.1, the within replication data can be highly correlated. In that section, it was mentioned that standard confidence intervals based on the regular formula for the sample variance

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

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

$$\text{Var}(\bar{X}) = \frac{\gamma_0}{n} \left[1 + 2 \sum_{k=1}^{n-1} \left(1 - \frac{k}{n}\right) \rho_k \right]$$

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:

$$E[S^2/n] = \frac{\gamma_0}{n} \left[1 - \frac{2R}{n-1} \right]$$

where

$$R = \sum_{k=1}^{n-1} \left(1 - \frac{k}{n}\right) \rho_k$$

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:

$$\text{Bias} = E[S^2/n] - \text{Var}(\bar{Y}) = \frac{-2\gamma_0 R}{n-1}$$

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 over estimates 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

5. Statistical Analysis for Infinite Horizon Simulation Models

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 $Var(\bar{X})$.

The method of batch means provides a way to develop (at least approximately) an unbiased estimator for $Var(\bar{X})$. Assuming that you have a series of data point, the method of batch means method divides the data into subsequences of contiguous batches:

$$\begin{aligned} &\underbrace{X_1, X_2, \dots, X_b}_{batch1} \cdots \underbrace{X_{b+1}, X_{b+2}, \dots, X_{2b}}_{batch2} \cdots \\ &\underbrace{X_{(j-1)b+1}, X_{(j-1)b+2}, \dots, X_{jb}}_{batchj} \cdots \underbrace{X_{(k-1)b+1}, X_{(k-1)b+2}, \dots, X_{kb}}_{batchk} \end{aligned}$$

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,

$$\bar{X}_j(b) = \frac{1}{b} \sum_{i=1}^b X_{(j-1)b+i}$$

Each of the batch means are treated like observations in the batch means series. For example, if the batch means are re-labeled 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:

$$\bar{Y}(k) = \frac{1}{k} \sum_{j=1}^k Y_j$$

$$S_b^2(k) = \frac{1}{k-1} \sum_{j=1}^k (Y_j - \bar{Y})^2$$

$$\bar{Y}(k) \pm t_{\alpha/2, k-1} \frac{S_b(k)}{\sqrt{k}}$$

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 towards 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 assess 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 have been developed that will automatically batch the data as it is collected, see for example (Fishman and Yarberry, 1997), (Steiger and Wilson, 2002), and Banks et al. (2005). has its own batching algorithm. The batching algorithm is described in Kelton et al. (2004) page 311. See also (Fishman, 2001) page 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):

$$C = \sqrt{\frac{k^2 - 1}{k - 2}} \left[\hat{\rho}_1 + \frac{[Y_1 - \bar{Y}]^2 + [Y_k - \bar{Y}]^2}{2 \sum_{j=1}^k (Y_j - \bar{Y})^2} \right]$$

$$\hat{\rho}_1 = \frac{\sum_{j=1}^{k-1} (Y_j - \bar{Y})(Y_{j+1} - \bar{Y})}{\sum_{j=1}^k (Y_j - \bar{Y})^2}$$

The hypothesis is rejected if $C > z_\alpha$ for a given confidence level α . If the batch means do not pass the test, *Correlated* is reported for the half-width on the statistical reports.

5.4.1. Performing the Method of Batch Means

Performing the method of batch means in is relatively straight forward. 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 5.1, i.e. $n_0 = 10$, $T_e = 30000$, $T_w = 3000$. Thus, your starting run length would be $n_0 \times T_e = 10 \times 30,000 = 300,000$ and the warm period will be $n_0 \times T_w = 30,000$. For these setting, the results shown in Figure 5.25 are very close to the results for the replication-deletion example.

Waiting Time	Average	Half Width
Seize Server.Queue	1.6225	0.061752921
Other		
Number Waiting	Average	Half Width
Seize Server.Queue	1.6241	0.064281088

Figure 5.25.: Initial batch means results.

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.

$$n \cong n_0 \left(\frac{h_0}{h} \right)^2$$

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 5.26. The resulting values for the number of batches formed for the waiting times and the size of the batches are given in Figure 5.27. Using this information in the half-width based sample size formula with $n_0 = 32$, $h_0 = 0.06$, and $h = 0.02$, yields:

$$n \cong n_0 \frac{h_0^2}{h^2} = 32 \times \frac{(0.06)^2}{(0.02)^2} = 288 \text{ batches}$$

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 5.26.: OUTPUT Statistics to get number of batches and batch size.

Output		Value
BSforQT		8192.00
NBforQT		32.0000

Figure 5.27.: Results for number of batches and batch size.

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 $(2,359,296 + 30,000 = 2389296)$. Re-running the simulation yields the results shown in Figure 5.28. 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 is easy to do by supplying the appropriate expression within the Terminating Condition field on the Run Setup > Replication Parameters dialog.

Figure 5.29 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

5. Statistical Analysis for Infinite Horizon Simulation Models

Waiting Time	Average	Half Width
Seize Server.Queue	1.6403	0.017292365
Other		
Number Waiting	Average	Half Width
Seize Server.Queue	1.6422	0.018253294

Figure 5.28.: Batch means results for fixed sample size.

terminate when the half-width criteria is met. The batching algorithm computes the value of $\text{THALF}(\text{Tally ID})$ after sufficient data has been observed. This expression can be expanded to include other performance measures in a compound Boolean statement.

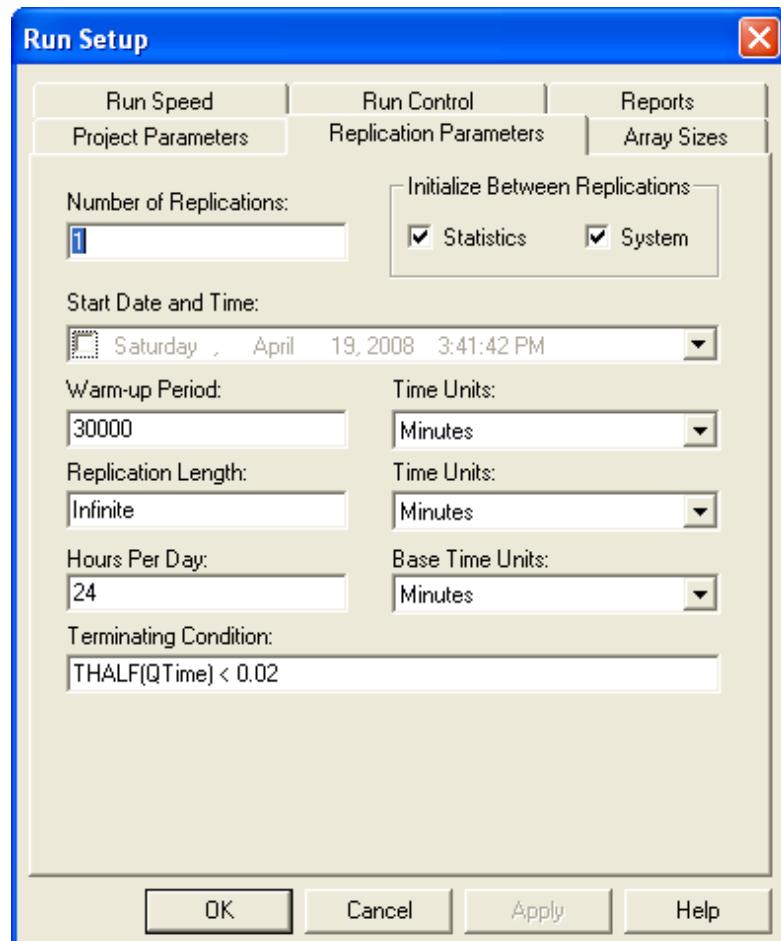


Figure 5.29.: Sequential sampling using terminating condition.

The results of running the simulation based on the sequential method are given in Figure 5.30. 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).

Waiting Time	Average	Half Width
Seize Server.Queue	1.6366	0.019929123
Other		
Number Waiting	Average	Half Width
Seize Server.Queue	1.6384	0.021182914

Figure 5.30.: Results for infinite horizon sequential sampling method.

Once the warm up period has been analyzed, performing infinite horizon simulations using the batch means method is relatively straight forward. A disadvantage of this 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. 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 is 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 in (Law, 2007).

When performing an infinite horizon simulation analysis, we are most interested in the estimation of long-run (steady state) performance measures. In this situation, it can be useful to apply analytical techniques such as queueing theory to assist with determining whether or not the model is producing credible results. Even in the case of a finite horizon simulation, the

5. Statistical Analysis for Infinite Horizon Simulation Models

steady state performance results from analytical models of queueing and inventory systems can be very helpful in understanding if the results produced by the simulation model make sense. In the next section, we apply the results of analytical queueing models from Appendix C to the simulation of a small manufacturing system in order to check the results of a simulation model. Being able to verify and validate a simulation model is a crucial skill to get your simulation models used in practice.

5.5. Applying Queueing Theory Results to Verify and Validate a Simulation

In this section, we overview some verification and validation methods to apply when developing and testing a simulation model.

Verification is about checking if the model meets the desired output. In other words, does the model work correctly as designed? In verifying a model, you select a sample of test cases with known results and show that the model produces the expected output across the test cases. It is important to select the test cases in an unbiased manner and that the test cases encompass a large portion of the space that will be relevant when using the model. The test cases form the standard that the model must meet and thus are critical in convincing others that the verification process was of high quality.

Validation is whether or not the model meets its intended purpose. When validating a model, we are trying to build credibility that the model can be used as intended. In other words, that the model adequately represents the system for its intended purpose. Obviously, verification is a necessary but not sufficient step in validation. Validation is predicated on how the model is intended to be used and what it is supposed to represent.

In the following, we will illustrate a basic approach to verifying and validating the results of a simulation model by using some analytical approximations based on queueing theory and common sense. Let's start with an example system to be simulated.

Example 5.1(Small Manufacturing System). A small manufacturing system produces parts. The parts arrive from an upstream Poisson process with a rate of arrival of 1 part every 5 minutes. All parts that enter the system must go through a preparation station where there are 2 preparation workers. A part requires only 1 of the 2 workers during preparation. The preparation time is exponentially distributed with mean of 8 minutes. If all the workers at the preparation station are busy, then the part waits for the next available preparation worker.

After preparation, the parts are processed on two different production lines. There is a 30% chance that the parts are built on line 1 and a 70% chance that they go to line 2. Line 1 has a build station staffed by 2 workers. Line 2 has a build station staffed by 3 workers. The time to build a

5.5. Applying Queueing Theory Results to Verify and Validate a Simulation

part on line 1 is triangularly distributed with a (min = 2, mode = 6, max = 8) minutes. The time to build a part on line 2 is triangularly distributed with a (min = 3, mode = 6, max = 7) minutes. The build operation on the part only requires 1 of the workers available at the station. If all the workers at the station are busy, then the part waits for the next available worker.

After the parts are built, they go to a packaging station. The packaging station is staffed by 2 workers. Only 1 of the 2 workers is needed during packaging. If all the workers at the packaging station are busy, then the part waits for the next available packaging worker. The time to individually wrap a part is exponential with a mean of 2 minutes. After each individual part is wrapped, the worker fills a box with packing peanuts, and places the part into a box for shipping. The time to fill the box with peanuts and seal the box is uniformly distribution between 1 and 2 minutes. After the packaging, the box is placed on a pallet building machine. Once 10 boxes accumulate, the pallet machine wraps the pallet. The wrapping time takes 2 minutes. After the wrapping is completed, the pallet leaves the system.

The production team believes that a batch run of production over 30 days of operation is the best way to produce these parts. Thus, at the beginning of a month there are currently no parts in production. Simulate the system for 100 replications of one month (30 days) of operation. The base time unit for the simulation should be in minutes. We want to simulate this system in order to measure the following performance metrics with some high degree of confidence.

Performance Measures

- System time for parts regardless of which build line prior to palletizer
 - Prob. that the system time before palletizing of a part is less than 60 minutes
 - Utilization of preparation workers
 - Utilization of packaging workers
 - Utilization of line 1 build workers
 - Utilization of line 2 build workers
 - Utilization of the palletizer
 - Total number of parts produced in 30 days
 - Total number of pallets completed in 30 days
-

Suppose you build a simulation model for this situation and get the following results. The building of the model is left as an exercise for the reader.

Performance Measures	Average	Half-width
System time for parts regardless of which build line prior to palletizer	31.73	0.44
Prob. that the system time before palletizing of a part is less than 60 minutes	0.89	0.01
Utilization of preparation workers	0.80	0.00
Utilization of packaging workers	0.35	0.00

5. Statistical Analysis for Infinite Horizon Simulation Models

Performance Measures	Average	Half-width
Utilization of line 1 build workers	0.16	0.00
Utilization of line 2 build workers	0.25	0.00
Utilization of the palletizer	0.039	0.00
Total number of parts produced in 30 days	8623.36	17.75
Total number of pallets completed in 30 days	861.85	1.77

How do you know if these results seem reasonable? You can use basic queueing analysis and some simple math to help verify and validate your results.

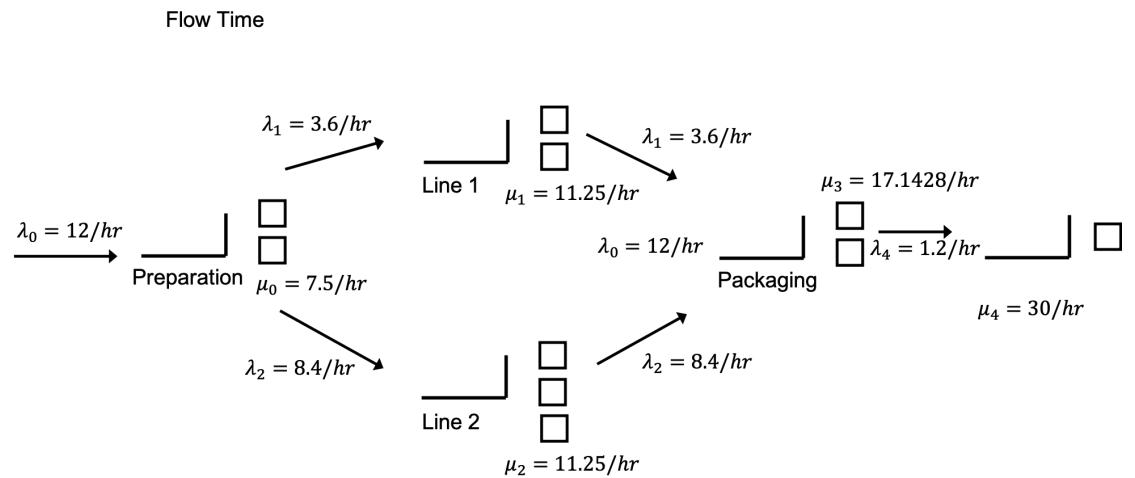


Figure 5.31.: Small manufacturing system with flow rates

By examining Figure 5.31 and using a little flow analysis, we can get a pretty good approximation of the results. In the figure, we see each station (preparation, line 1, line 2, packaging, and the palletizer). The flow rate of the parts to each of the stations has also been noted. In what follows each of the stations will be analyzed (individually) and then the overall system will be analyzed using some basic queueing analysis using the results of Appendix C.

The approach is based on determining the rate of flow between the stations and using simple queueing formulas. Let's start with the preparation station.

5.5.1. Analyzing the Preparation Station

Let λ_0 be the arrival rate to the preparation station. Let μ_0 be the service rate at the preparation station. This is the rate at which an individual preparation worker works. Let $MTBA_0$ be the mean time between arrivals for the preparation station and let $E[ST_0]$ be the mean service time at the preparation station. Clearly from the problem description, we have:

5.5. Applying Queueing Theory Results to Verify and Validate a Simulation

$$\lambda_0 = \frac{1}{\text{MTBA}_0} = \frac{1}{5 \text{ min}} \times \frac{60 \text{ min}}{1 \text{ hour}} = 12/\text{hour}$$

$$\mu_0 = \frac{1}{E[\text{ST}_0]} = \frac{1}{8 \text{ min}} \times \frac{60 \text{ min}}{1 \text{ hour}} = 7.5/\text{hour}$$

Based on queueing theory, we have that the utilization, ρ , is:

$$\rho = \frac{\lambda}{c * \mu} = \frac{E[\text{ST}]}{c * \text{MTBA}}$$

where c is the number of servers at the station. For the preparation station, we have:

$$\rho_0 = \frac{\lambda_0}{c\mu_0} = \frac{12}{2 \times 7.5} = 0.8$$

From the simulation output results, we see that the utilization of the simulation is exactly that as predicted from the basic flow analysis. Let's examine the build lines.

5.5.2. Analyzing the Build Lines

In this case, we need to determine the arrival rate to each of the lines. First, we assume that the rate of arrivals out of the preparation station will be the same as the rate of arrivals into the preparation station. Since there is a probabilistic split of the flow to the lines, we can split the rate of parts flowing to line 1 and line 2 based on the probability of selecting the lines. Thus, we have that each line experiences the input rates as follows.

- Let λ_1 be the arrival rate to the line 1 station and let λ_2 be the arrival rate to the line 2 station.
- Let p_1 be the probability that line 1 is selected and let p_2 be the probability that line 2 is selected.

Thus, $\lambda_1 = p_1 \times \lambda_0 = 0.3 \times 12 = 3.6$ per hour and $\lambda_2 = p_2 \times \lambda_0 = 0.7 \times 12 = 8.4$ per hour. Now, we need to get the mean service time for the build lines. Because the service time distribution is triangular, we have that the mean of the triangular random variable, X , is:

$$E[X] = \frac{(\text{min} + \text{mode} + \text{max})}{3}$$

Therefore, we can compute the service rate for each of the lines, where ST_1 and ST_2 represent the service time random variable for line 1 and 2, respectively, as follows:

$$\mu_1 = \frac{1}{E[\text{ST}_1]} = \frac{(2 + 6 + 8)}{3} = 5.33 \text{ min} \times \frac{60 \text{ min}}{1 \text{ hour}} = 11.25/\text{hour}$$

5. Statistical Analysis for Infinite Horizon Simulation Models

$$\mu_2 = \frac{1}{E[ST_2]} = \frac{(3 + 6 + 7)}{3} = 5.3\bar{3} \text{ min} \times \frac{60 \text{ min}}{1 \text{ hour}} = 11.25/\text{hour}$$

And, then we can compute the utilization for each of the lines.

$$\rho_1 = \frac{\lambda_1}{c\mu_1} = \frac{3.6}{2 \times 11.25} = 0.16$$

$$\rho_2 = \frac{\lambda_2}{c\mu_2} = \frac{8.4}{3 \times 11.25} = 0.24\bar{8}$$

Again, these results, match very closely the results from the simulation model. We can also approximate the results for packaging in the same manner.

5.5.3. Analyzing the Packaging Station

Again, we need to compute the incoming flow rate to the packaging station in order to compute the utilization at the station. Let λ_3 be the arrival rate to the packaging station. If we assume that everything that arrives to each of the two build lines eventually gets to packaging, we can see that $\lambda_3 = \lambda_1 + \lambda_2 = \lambda_0 = 12$ per hour.

The mean service time at the packaging station is the sum of the mean wrapping process time and the mean box filling time. The wrapping process has an exponential distribution with a mean of 2 minutes while the filling process has a continuous uniform distribution on the range of 1 to 2 minutes. Thus, using the expected values for these respective distributions, we have:

$$E[ST_w] = 2 \text{ min}$$

$$E[ST_f] = 1.5 \text{ min}$$

Let ST_3 be the service time at the packaging station. Because the time in service at the packaging station include both processing times, we have a utilization calculation as follows:

$$E[ST_3] = E[ST_w] + E[ST_f] = 3.5 \text{ min}$$

$$\mu_3 = \frac{1}{E[ST_3]} = \frac{1}{3.5 \text{ min}} \times \frac{60 \text{ min}}{1 \text{ hour}} = 17.1428 \text{ per hour}$$

$$\rho_3 = \frac{\lambda_3}{c\mu_3} = \frac{12}{2 \times 17.1428} = 0.35$$

Again, the results match very closely to the simulation results. With some additional thinking, we can even approximate the utilization of the palletizer.

5.5.4. Analyzing the Palletizing Station

Assuming that all parts that arrive to packaging also arrive to the palletizer, we have a rate of arrivals to the palletizer of 12 per hour. However, we have to wait until there are 10 parts before they are sent to the palletizer. The rate of 12 per hour is a mean time between arrivals of 1 part every 5 minutes. To get 10 parts, it will take on average 50 minutes. Thus, the mean time between arrivals to the palletizer is 50 minutes. Let λ_4 be the arrival rate to the palletizer. Thus, the arrival rate to the palletizer is

$$\lambda_4 = \frac{1}{\text{MTBA}_4} = \frac{1}{50 \text{ min}}$$

Thus, the utilization for the palletizer should be:

$$\rho_4 = \frac{\lambda_4}{c\mu_4} = \frac{E[ST_4]}{c * \text{MTBA}_4} = \frac{2}{1 \times 50} = 0.04$$

Again, this matches quite well with the simulation results. In general, as long as $\lambda < c\mu$ the approximation of utilization will be quite accurate.

5.5.5. Analyzing the Total System Time

We can go even further and approximate the total time spent in the system. If there is no waiting in any of the queues, the approximation is quite simple. We just need to total the time spent in service for the parts. Let $E[ST]$ be the total time spent in service. To compute this value, we can simply add up all of the mean service times for the stations visited. In the case of line 1 and line 2, we can take the weighted average based on the probability that the line is selected as follows:

$$E[ST] = E[ST_0] + p_1 E[ST_1] + p_2 E[ST_2] + E[ST_3] + E[ST_4]$$

$$E[ST] = 8 + 0.3 \times 5.3\bar{3} + 0.7 \times 5.\bar{3}\bar{3} + 3.5 + 2.0 = 18.8\bar{3} \text{ minutes}$$

This can be easily tested within the simulation model by setting all the resource capacities to “infinite”. If you do this, the results will indicate that the expected time in the system is 18.8951 ± 0.06 with 95% confidence. Clearly, these results are close enough to provide even more credibility to the simulation results.

By using additional queueing theory, we can approximate total time spent at each station by approximating the time spent within the queues and then adding up the total time. This will require additional approximations. For a random variable, X , the squared coefficient of variation is given by:

5. Statistical Analysis for Infinite Horizon Simulation Models

$$c_X^2 = \frac{\text{Var}(X)}{(E[X])^2}$$

Whitt (1983) suggests the following approximation for the queuing time in a GI/G/c queue, where GI is some general independent inter-arrival distribution, G is some general service distribution and c is the number of servers. The approximation has the following form:

$$W_q(\text{GI}/G/c) \cong \left(\frac{c_a^2 + c_s^2}{2} \right) W_q(M/M/c)$$

Where c_a^2 and c_s^2 represent the squared coefficient of variation of the inter-arrival time distribution and the service time distribution respectively and $W_q(M/M/c)$ is the expected waiting time in a M/M/c queue.

The main difficulty with applying this approximation is determining the variance of the inter-arrival time distribution. In what follows, we will simplify the approach even further and assume that the inter-arrival process at each station is a Poisson process. Thus, the inter-arrival distribution will be assumed to be exponential. Since the variance of an exponential distribution is the square of its mean, the squared coefficient of variation for all the arrival processes will be $c_a^2 = 1$. We will just need to compute the squared coefficient of variation for each of the service time distributions associated with each station. To determine the waiting time in queue, we will need the waiting time in queue equations for the M/M/c queueing model. The following table summarizes the formulas for the cases of 1, 2, and 3 servers.

Table 5.5.: Formulas for expected waiting time for M/M/c queue $\rho = \lambda/c\mu$

$c = \# \text{ servers}$	$W_q = \text{Expected Waiting time in Queue}$
1	$\frac{\rho^2}{\lambda(1-\rho)}$
2	$\frac{2\rho^3}{\lambda(1-\rho^2)}$
3	$\frac{9\rho^4}{\lambda(2+2\rho-\rho^2-3\rho^3)}$

We can now tabulate the waiting time in the queue for each of the stations. By using the variance associated with each service time distribution, as well as the mean, we can compute the squared coefficient of variation for the service time distribution for each station. The following table summarizes the results. The calculations are left to the reader.

Table 5.6.: Summary queueing results for each station

Station	c	$\rho = \frac{\lambda}{c\mu}$	$W_q(M/M/c)$	c_a^2	c_s^2	$\frac{c_a^2 + c_s^2}{2}$	$W_q(\text{GI}/G/c)$
Preparation	2	0.8	14.22	1	1	1	14.22
Line 1 Build	2	0.16	0.0420	1	0.0547	0.5274	0.0221
Line 2 Build	3	0.248	0.0722	1	0.0254	0.5127	0.0370

5.5. Applying Queueing Theory Results to Verify and Validate a Simulation

Station	c	$\rho = \frac{\lambda}{c\mu}$	$W_q(M/M/c)$	c_a^2	c_s^2	$\frac{(c_a^2 + c_s^2)}{2}$	$W_q(\text{GI}/G/c)$
Packaging	2	0.35	0.4886	1	0.3333	0.6666	0.3259
Palletizing	1	0.04	0.0722	1	0	0.5	0.0361

From these results, we can tabulate an approximation for the total time in the system. Let $E[TW_q]$ be the total expected waiting time of a part and $E[T]$ be the expected total time spent in the system for a part. Clearly, we have that, $E[T] = E[TW_q] + E[ST]$, where $E[ST]$ is the total time spent in service for a part. Using the results of the table and taking a weighted average across the waiting time for the two build lines, we can see that the total expected waiting time in queue is:

$$E[TW_q] = 14.22 + (0.3) * (0.420) + (0.7) * (0.0722) + 0.4886 + 0.0722 = 14.957$$

Thus, the total expected time spent in the system should be approximately, $E[T] = E[TW_q] + E[ST] = 18.83 + 14.957 = 33.79$ minutes. We can compare this value to the simulation result (31.73 ± 0.44) with 95% confidence.

Based on this comparison, we can see that the approximation is very good. Again, this should add a great deal of credibility to the reported results. You might think that, in general, simulation models will be “too complex” for such validation methods. Well, sure, I agree that there can be very complex simulation models, but that does not mean that parts of the model would not lend themselves to a simple analysis as demonstrated here. If the individual parts of a simulation model can be validated. Then, as long as we believe that when putting the parts together to form the entire model, it is still valid, then why wouldn’t the overall results not be valid? By verifying and validating individual components of the model, we can build up the credibility in the overall model.

5.5.6. Other Issues for Verification and Validation

As previously noted, verification is about checking if the model or algorithm produces its desired output. Verification is an issue of software quality control. Since computer simulation relies on representing a system within software, we must ensure that we produce software (code) that is error-free. While it is beyond the scope of this book to fully discuss quality control issues within software development, we will note two major approaches that should be applied to simulation model development: 1) testing and 2) debugging.

The test cases should be designed from simple tests to more complicated tests. Test cases can be designed to test individual components of the model. If each component passes its test, then testing all components together should occur. For a stochastic simulation, testing can be performed using statistical tests over the set of test cases that indicate with a high level of confidence that the model produces results that are not significantly different than the expected

5. Statistical Analysis for Infinite Horizon Simulation Models

output. In addition, you might show that the model converges to the expected result. Here, error and relative error, their statistical properties, etc. are important metrics to use to provide overall confidence in the verification process.

Testing a simulation model begins by developing the model *in stages*. You should first identify the key modeling issues and develop pseudo-code to address those issues. Then, you should map out a logical development process by which the components of the model can be constructed as independently of each other as possible. For example, for the small manufacturing system, the first stage of the model building process would be to simulate the preparation station. We can use DISPOSE modules to dispose of those parts that leave the preparation station and analyze whether or not the preparation station is working correctly. You should identify parts of the system that can be analyzed with very little dependence on the other components. If we only want to check if the logic of the component is working correctly, we can use a simple CREATE module that causes a simpler deterministic arrival pattern and step through the model logic. For example, create a single entity and send it through the model. Does it do what you intended? Force the entity to take a particular path through the system by setting appropriate attributes or changing appropriate DECIDE modules. Does the entity do what it is supposed to do for the test? As noted in Example 5.1 another approach is to set the capacity of all resources to infinity and run the model. Then, you can check if the basic flow time is as expected without any queueing. A staged development process that starts with pseudo-code and tests each stage is critical to ensuring that the model works as intended. Start small. Build small. Test small. And, then integrate the components.

The second recommended method of verification is debugging the model. Use the debugging capability of your software and step through the model. Check edge cases and debug the model. Run the model with watches on variables that stop the model if an invalid condition occurs, trace output through the debugger or by printing out data, watch well designed animation examples that have expected behavior. Animation can be essential to showing that your model is working as intended. Add animation elements that help you determine if your model logic is working. Regardless of whether or not your simulation requires a beautiful animation, you should utilize the basic animation capabilities to verify that your model works. Document what you did for the debugging. State what you did and what you found. The fact that you performed these verification steps is the essential first step in validating your model. Validation is whether or not the model meets its intended purpose. Validation cannot be accomplished unless verification steps have occurred.

Validation cannot be done without convincing the user that the key conceptual elements of the system have been adequately represented in the model. This includes the modeling of the inputs to the system and how the conceptual model has been translated to a computerized representation. This is why documenting the input models and the conceptual model is so important. The old axiom, garbage in equals garbage out is pertinent here. The user must believe that the inputs have been well-modeled. The user must understand what is being modeled.

It is also important that everyone agrees on how the model is intended to be used. If there is no agreement on the conceptual representation or on the model's use, then validation will be unsuccessful. Validation is predicated on the model representing the system. Thus, validation

often involves comparing the output of the model to the output of the real system. If the user of the model cannot tell the difference between model output and real system output, then the model's representational power is quite high. If you have data from the real system, especially system output performance measures (e.g. throughput, system time, etc.), then you can compare the results of the simulation model to the actual results from the system. You can perform statistical tests to show that the results from the simulation model are not statistically different from the actual system. Again, a staged approach can be used. Do not feel that you have to validate the whole model against the whole system. Perhaps you can collect data on some critical sub-component of the system being modeled. You can then compare the results of your simulation of that critical sub-component to the actual results from that sub-component. Validating each sub-component goes a long way to validating the overall model. It may be easier to collect data from the actual system in stages or in these smaller components. Besides checking if the output matches the actual system output, you should check if the simulation model behaves as expected. The output of the model should react in the same manner as that expected for the real system or the conceptual understanding of how the system works. If you increase the arrival rate, does congestion go up? If not, it is likely something may be misrepresented within your model.

In many situations, the real system does not exist yet. Thus, you will not have real data for validation purposes. This is often the case when simulation is being used to design a system. In such situations, you may be able to approximate the expected performance via "hand" calculations. In this context, you can and should approximate the performance of the simulation model using analytical methods, such as queuing theory. This can provide bounds on the expected performance and add to the credibility of the results. In addition, even if the whole system may not yet exist, perhaps sub-components of the real system exist. If so, you can test output from the sub-components to the associated sub-components of your model. Do not think like you have validate the whole model! Think about how you can validate parts of the model, with the validation of each part contributing to the validation of the whole.

Carefully documenting your conceptual model and your assumptions is also an important task for validation. If the user of your model does not understand it through its careful documentation, then they are less likely to believe the results from the model. Performing sensitivity analysis on a set of representative test cases can build credibility in the usage of the model. Validation is ultimately about believability. Does the user believe the results of the model? Validation is whether or not the model/algorithm meets its intended purpose. Verification is about checking if the model/algorithm meets the desired output. You must perform these functions when building a simulation model.

5.6. Summary

Based on the discussion in this chapter and in Chapter 3, we have introduced many of the statistical aspects of simulation modeling and analysis. The first issue that you face is determining whether or not the simulation has a finite or an infinite horizon. Based on that determination,

5. Statistical Analysis for Infinite Horizon Simulation Models

you must determine how much to sample in terms of the number of replications or in the case of batch means the length of the simulation run. We have discussed using the normal approximation method and the half-width ratio method with Chapter ??ch3). In the case of an infinite horizon simulation the same concepts reappear when using the method of replication deletion or when performing a batch means analysis. After determining your statistical environment you can better plan out how you will execute the simulation to make decisions. The same techniques that were discussed in Section 4.4 of Chapter 4 can still be used when comparing systems that have an infinite horizon. If you use the method replication-deletion, then there no conceptual difference in comparing of two design alternatives. If you choose to use the batch means method, then the approach is to utilize the batch means from each of the two design configurations as if they were random samples. Unfortunately, batching complicates the use of common random numbers. Thus, I recommend utilizing the method of replication-deletion when your infinite horizon simulation scenarios must be compared.

The next chapter will address more advanced modeling techniques for process modeling. Specifically, non-stationary arrivals, additional concepts in resource modeling, and other miscellaneous modeling techniques. At this point, you should be able to model a wide variety of interesting systems and perform the basic statistical analysis on those models that are required to make valid statistical conclusions.

5.7. Exercises

Exercise 5.1. The batch means method for analyzing 1 long simulation run is needed because within replication data are often (a) _____ and (b) _____.

Exercise 5.2. 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 don't account for these conditions. One method to mitigate this problem is to perform a (c)_____ analysis to determine an appropriate (d)_____ period.

Exercise 5.3. *True or False:* The concept of steady state implies that after a long enough time the system will not change with respect to time.

Exercise 5.4. 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.

Exercise 5.5. Figure 5.32 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 5 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.

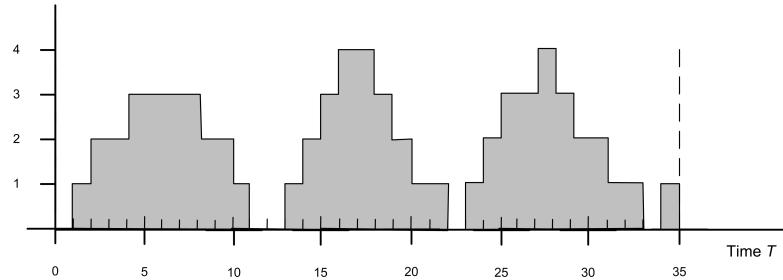


Figure 5.32.: Queue length sample path

Exercise 5.6. Using the supplied data set, draw the sample path for the state variable, $N(t)$. 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. 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.

t	0	2	4	5	7	10	12	15	20
$N(t)$	0	1	0	1	2	3	2	1	0

Exercise 5.7. 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 1 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 a Welch plot using 5 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)?
- 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.
- Use the method of replication deletion to develop a 95% confidence interval for the mean waiting time. Use your warm period from part (c). Compare the result with that of (a) and (e) and discuss.

Exercise 5.8. YBox video game players arrive at a two-person station for testing. The inspection time per YBox set is EXPO(10) minutes. On the average 82% of the sets pass inspection. The remaining 18% are 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 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.
-

Exercise 5.9. 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 . Include a warm up analysis of the system time and estimate the system time to within 2 minutes with 95% confidence.

Exercise 5.10. 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 2 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 6 parts in the preparation queue. Any parts that arrive to the system when there are 6 or more parts in the preparation queue cannot enter the system. These parts are shunted to a re-circulating conveyor, which takes 10 minutes to re-circulate the parts before they can try again to enter the preparation queue. Hint: Model the re-circulating 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 1 of 4 operators assigned to the build station. The

5. Statistical Analysis for Infinite Horizon Simulation Models

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 average exponentially distributed. The part remains in the tooling fixture after the first operation and proceeds to the second operation. There is 1 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 re-circulating conveyor.
- The average number of parts on the conveyor.

Perform a warm up analysis on the system time of a part regardless of type.

Exercise 5.11. Reconsider Exercise 5.10. 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 log-normally 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?

Exercise 5.12. A patient arrives at the Emergency Room about every 20 ± 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 ± 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 ± 25 minutes (stream 3) and are discharged.

CW patients initially receive a priority of 1 and are treated (when their turn comes) for 15 ± 14 minutes (stream 4); their priority is then increased to 2, they wait again until a doctor is free, receive 10 ± 8 minutes (stream 5) of final treatment, and are discharged.

An important aspect of this system is that patients that have already seen the doctor once compete with newly arriving patients that 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 Help files. In addition, we assume that the doctors are interchangeable. That is, it does not matter which of the 2 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.

Measure the average queue length of NIA patients from arrival to first seeing a doctor. What percent have to wait to see the doctor for the first treatment? Report statistics on the initial waiting time for NIA patients. What percent wait less than 5 minutes before seeing a doctor for the first time? Report statistics on the time in system for the patients. Report statistics on the remaining time in system from after the first treatment to discharge, for all patients. 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.

Exercise 5.13. 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 $j = 0, 1, 2$, etc, 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.

Exercise 5.14. An airline ticket office has two ticket agents answering incoming phone calls for flight reservations. In addition, six callers can be put on hold until one of the agents is available to take the call. If all eight 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.

In addition, the ticket office has 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 average for the phone call, still exponentially distributed. Regular customers still take on 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.

Exercise 5.15. Consider a system having 6 machines tended by 2 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 first come first served queue with no priority given to either type of stoppage. Simulate this system for 10000 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.

Exercise 5.16. 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 inter-arrival 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 a model for this system using the STATION and ROUTE modules.

- a. Run the simulation for exactly 20000 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 queueing 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).
-

Exercise 5.17. 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 2 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 setup the part on the machine. After the machine is setup, 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 setups and part removals. The operator attends to the parts in a first come, first served manner. The times for the preparation machines are given in the table below according to a triangular distribution with the provided parameters:

Prep 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 4 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

5. Statistical Analysis for Infinite Horizon Simulation Models

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 prep 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?
-

Exercise 5.18. 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 log-normally 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. Figure 5.33 provides an illustration of the system.

(This problem is based on an example on page 209 and continues on page 217 of (Pegden et al., 1995). Used with permission)

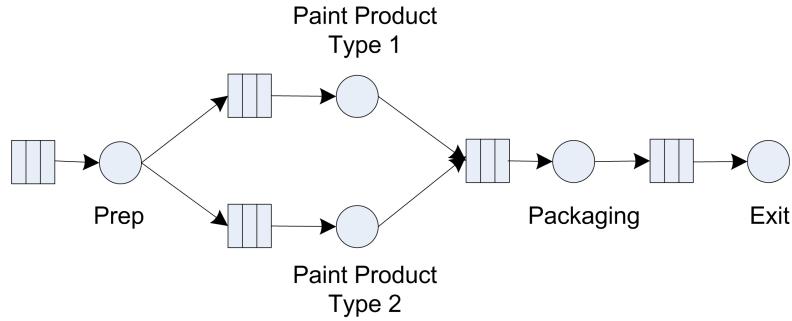


Figure 5.33.: Simple painting flow line

Suppose that statistics on the part flow time, i.e. 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 percent 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 log-normally 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?

Exercise 5.19. Consider the small manufacturing system of Example 5.1 build a simulation model that produces the results shown in Section 5.5. What happens if the arrival rate of the parts to the system increases by 50%?

6. Modeling Systems with Advanced Process Concepts

LEARNING OBJECTIVES

- To be able to model non-stationary arrivals using arrival schedules.
- To be able to model the staffing/scheduling and failure/repair of resources using resource capacity schedules and failures.
- To be able to capture statistics over specific periods of time.
- To be able to model balking and reneging within queuing situations.
- To be able to model situations involving holding and signaling entities.
- To be able to model situations involving picking stations, picking up entities into groups, dropping off entities from groups, and generic station modeling.

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 systems. Because non-stationary arrivals affect how statistics should be interpreted, we will also study how to collect statistics over specific periods of time.

The chapter also covers some useful modeling situation involving how entities interact within Arena. The first situation that we will see is that of reneging from a queue. This will introduce the SEARCH and REMOVE modules within Arena. Then, we will study advanced concepts involving how to signal entities based on general conditions within the system. In my opinion, this is one of the most useful modeling constructs that you will need within practice. Finally, the chapter covers some miscellaneous (but useful) topics. Specifically, as entities move through the model, they often are formed into groups. Arena's constructs for grouping and un-grouping entities will be presented. Grouping entities is similar in some respects to batching entities, but we will see that Arena's grouping/ungrouping functionality can lead to additional modeling flexibility. Let's get started by looking at how time dependent arrivals can be generated.

6.1. 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's 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 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 performed 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 the 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 as demonstrated in Section B.3.1. Even if you do not divide time into logical intervals, it is good practice to record the date and 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 λ per hour.

As a reminder, λ represents the mean arrival rate or the expected number of customers per unit time. Suppose that the expected arrival rate is five per hour. This does not mean that the pharmacy will get five customers every hour, but rather that on 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, λ 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's 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 inter-arrival 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, $E[X] = 12$, $Var[X] = 4$, 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. This can easily be accomplished by holding the distributions within an arrayed expression and using a clock entity to determine which period of time is active and indexing into the arrayed expression accordingly. 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 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

6. Modeling Systems with Advanced Process Concepts

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 6.1.

Table 6.1.: Example arrival counts

Interval	Mon	Tue	Wed	Thurs	Fri	Sat	Sun
12 am – 6 am	3	2	1	2	4	2	1
6 am - 9 am	6	4	6	7	8	7	4
9 am - 12 pm	10	6	4	8	10	9	5
12 pm - 2 pm	24	25	22	19	26	27	20
2 pm - 5 pm	10	16	14	16	13	16	15
5 pm - 8 pm	30	36	26	35	33	32	18
8 pm - 12 am	14	12	8	13	12	15	10

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's accept these groups as appropriate for this situation. Looking at the intervals, you see that more arrivals tend to occur from 12 pm to 2 pm and from 5 pm to 8 pm. As discussed in Section B.3.1, we could test for this non-stationary 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 non-homogeneous Poisson process as the arrival process.

Consider the first interval to be 12 am to 6 am; 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 use to perform 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 than 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 we illustrated in Section B.3.1, one method of doing this is to perform a contingency table test.

Let's 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

6.1. Non-stationary Processes

can calculate the average arrival rate for each interval and the rate per hour for each interval as shown in Table 6.2. In the table, 2.14 is the average of the counts across the days for the interval 12 am to 6 pm. 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 6.1 illustrates the time-varying behavior of the mean arrival rates over the course of a day.

Table 6.2.: Mean arrival rates for each time interval

Interval	Avg.	Length (hrs)	Rate per hour
12 am – 6 am	2.14	6	0.36
6 am - 9 am	6.0	3	2.0
9 am - 12 pm	7.43	3	2.48
12 pm - 2 pm	23.29	2	11.645
2 pm - 5 pm	14.29	3	4.76
5 pm - 8 pm	30.0	3	10
8 pm - 12 am	12.0	4	3.0

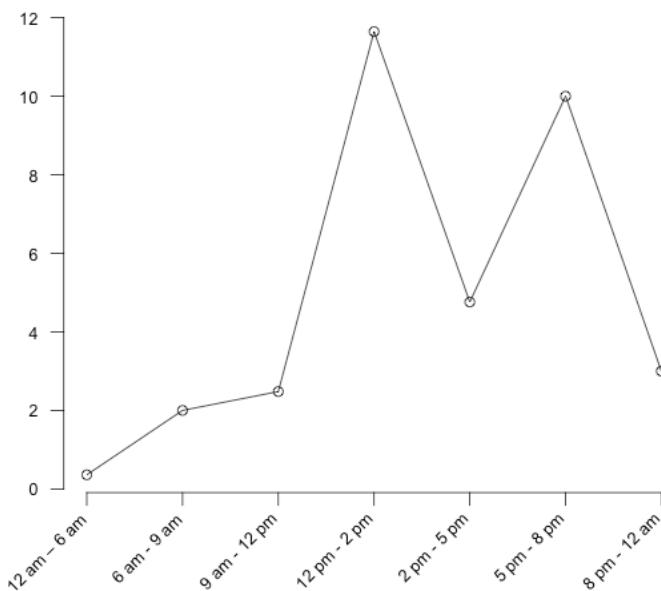


Figure 6.1.: Arrival rate per hour for time intervals

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 (Leemis and Park, 2006) or (Ross, 1997) for more of the theory underlying these

6. Modeling Systems with Advanced Process Concepts

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 inter-arrival 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. 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.

6.1.1. Thinning Method

For the thinning method, a stationary Poisson process with a constant rate, λ^* , and arrival times, t_i^* , is first generated, and then potential arrivals, t_i^* , are rejected with probability:

$$p_r = 1 - \frac{\lambda(t_i^*)}{\lambda^*}$$

where $\lambda(t)$ is the mean arrival rate as a function of time. Often, λ^* is set as the maximum arrival rate over the time horizon, such as:

$$\lambda^* = \max_t \{\lambda(t)\}$$

In the example, λ^* would be the mean of the 5 pm to 8 pm interval, $\lambda^* = 11.645$. The following pseudo-code for implementing the thinning method creates entities at the maximum rate and thins them according to which period is currently active.

```

CREATE entities, mean time between arrivals, EXPO(1/\lambda*)
DECIDE with chance, 1 - λ(t_i^*) / λ*
  IF accepted THEN
    Go to Label B
  ELSE
    DISPOSE
  END IF
END DECIDE
B: enter model and use arrival

```

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.

```

CREATE 1 logical entity
A: ASSIGN vPeriod = vPeriod + 1
DECIDED IF vPeriod < vNumPeriods THEN
  DELAY for period length
ELSE
  vPeriod = 0
END DECIDE
GO TO Label A:

```

One of the exercises in this chapter requires implementing the thinning algorithm in . The theoretical basis for why thinning works can be found in (Ross, 1997).

6.1.2. 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 re-scale 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 method in simulation. This method is available for piece-wise constant-rate functions in by using an Arrivals Schedule associated with a CREATE module. Let's see how to implement the example using an Arrivals Schedule.

The following is available in the file *PharmacyModelNSPP.doe* that accompanies this chapter. The first step in implementing a non-stationary Poisson process in is to define the intervals and compute the rates per hour as per Table 6.2. 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.

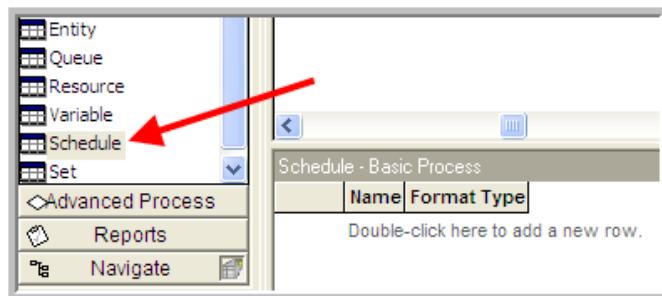


Figure 6.2.: SCHEDULE data module

Figure 6.2 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 text boxes for the SCHEDULE dialog follow:

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, months, etc.).

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 6.3.

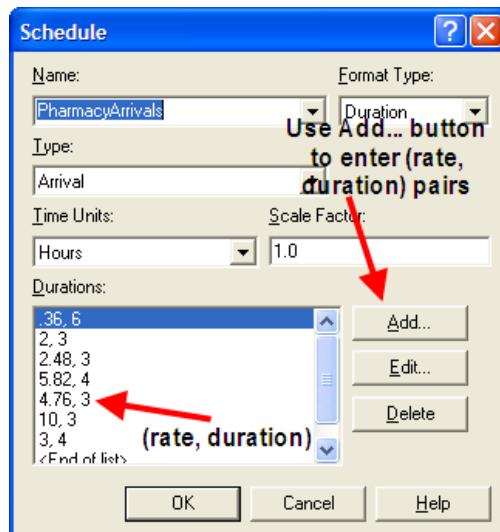


Figure 6.3.: SCHEDULE dialog

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 duration values. The duration values can also be entered using a spreadsheet like view.

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 6.4 where the Time Between Arrivals Type has been specified as Schedule with the appropriate Schedule Name. The schedule as in Figure 6.3 shows that the duration values 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 one 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.

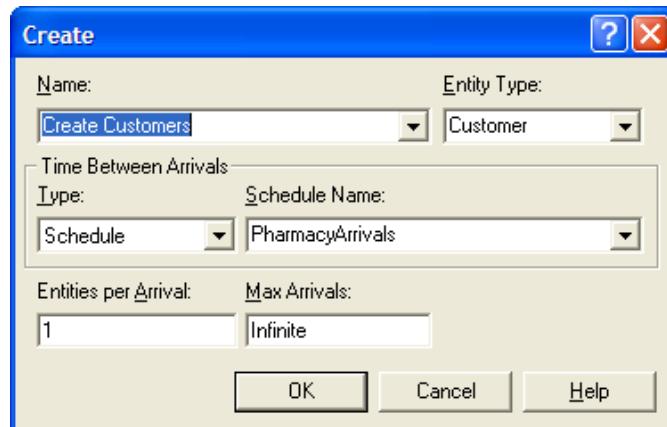


Figure 6.4.: CREATE module with schedule type

As indicated in this limited example, it is relatively easy to model a non-stationary Poisson arrival process within Arena. If the arrival process to the system is non-stationary, then it is probably a good idea to vary the resources available in they system according to the arrival pattern. The next section discusses how to extend the modeling of resources to include time varying behavior as well as the ability to fail at random points in time.

6.2. 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 two states: Busy or Idle. This section discusses the other two default states that permits for resources: Failed and Inactive.

6. Modeling Systems with Advanced Process Concepts

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 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 preempts 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:

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 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 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 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 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 environment (e.g. variable animation) to check the state of a resource.



How do we check if a resource is busy? You can use either of the two following expressions to check if a resource is busy: 1) STATE(Resource ID) == BUSY_RES, or 2) NR(Resource ID) > 0, where Resource ID is the name of your resource that you want to check.

The function STATE(Resource ID) returns the current state of the specified Resource ID as defined in the *Statesets* option for the resource. The STATE variable returns an integer number corresponding to the position within the specified Resource ID's associated stateset. It also may be used to assign a new state to the 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.

6.2.1. 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 2 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 6.5 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 1 pharmacist. Then, after the 4 hour delay, the capacity is increased back up to the 2 pharmacists. This is essentially how you utilize the ALTER block within a model. Please refer to 1 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 SMART file. In this system, customers arrive according to a Poisson process with a rate of 1 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's suppose that two, 480 minute shifts, of operation will be simulated. Assuming that the first shift starts at time zero, each shift is staffed with the same pattern, which varies according to the values shown in Table 6.3. With this information, you can easily setup the resource in to follow the staffing plan for the two shifts.

6. Modeling Systems with Advanced Process Concepts

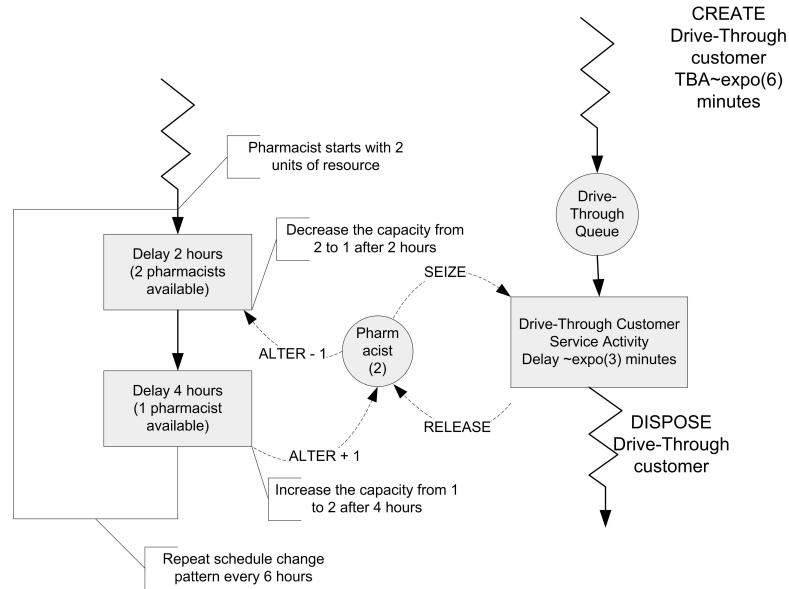


Figure 6.5.: Activity diagram for scheduled capacity Changes

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.

Table 6.3.: Example 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

Enter (value, duration) pairs value = capacity duration = time e.g. (4 units available for 1 hour)						
For durations						
Schedule - Basic Process	Name	Format Type	Type	Time Units	Scale Factor	Durations
1	Schedule 1	Duration	Capacity	Hours	1.0	4 rows
Double-click here to add a new row.						

Figure 6.6.: Resource schedule value, duration data sheet view

372

To setup the schedule, there are a number of schedule editing formats available. Figure 6.6 shows the schedule spreadsheet editor for this example. In the data sheet, 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), type (Capacity for resources), and time units (hours in this case). The scale factor field does not apply to capacity type schedules.

The Schedule dialog box is also useful in entering a schedule. The method of schedule editing is entirely your preference. Arena also allows schedules to be read in from Excel.

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 6.7. In addition, you must specify the rule that the resource will use if there is a schedule change 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 6.7. Let's discuss a bit more on how these rules function with an example.

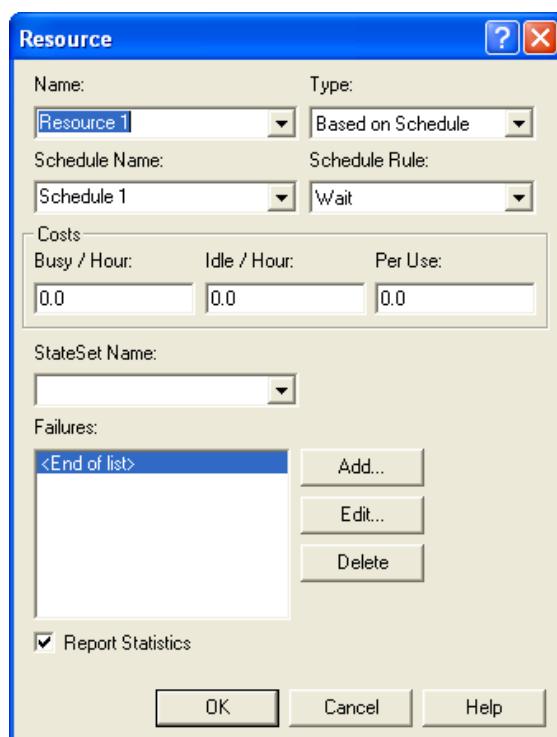


Figure 6.7.: RESOURCE module dialog with schedule capacity

Let's suppose that your simulation professor has office hours throughout the day, except from 12-12:30 pm for lunch. In addition, since your professor teaches simulation using , he or she

6. Modeling Systems with Advanced Process Concepts

follows Arena's rules for capacity changes. What happens for each rule if you arrive at 11:55 am with a question?

Ignore Case 1:

- You arrive at 11:55 am with a 10 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 5 minutes.
- You leave at 12:05 pm.
- Whether or not there are any students waiting in the hallway, the professor still starts lunch.
- At 12:30 pm 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.

Ignore Case 2:

- You arrive at 11:55 am 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 pm the professor gets up and takes down the lunch in progress sign and continues to answer your question.
- You leave at 12:40 pm

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's 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's examine what happens if the rule is Wait.

Wait Case 1:

- You arrive at 11:55 am 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 doesn't act on it, yet.
- Your professor continues to answer your question for the remaining service time of 5 minutes.
- You leave at 12:05 pm. 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 pm 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.

Wait Case 2:

- You arrive at 11:55 am 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 doesn't act on it, yet.
- Your professor continues to answer your question for the remaining service time of 40 minutes.
- You leave at 12:40 pm. 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 pm 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's consider the situation of the Preempt rule.

Preempt Case 1:

- You arrive at 11:55 am 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 pm.
- At 12:30 pm 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.

6. Modeling Systems with Advanced Process Concepts

- At 12:35 pm, 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 is essentially the same, with the student finally completing service at 12:55 pm. 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's run the example model. The final model including the animation is shown in Figure 6.8. 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 6.9. 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's take a more detailed look at how these quantities are computed.

6.2.2. 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 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:

$$\overline{NR} = \frac{1}{T} \int_0^T NR(t) dt$$

Let $MR(t)$ be the number of scheduled resource units at time t . Then, the time average number of scheduled resource units is:

$$\overline{MR} = \frac{1}{T} \int_0^T MR(t) dt$$

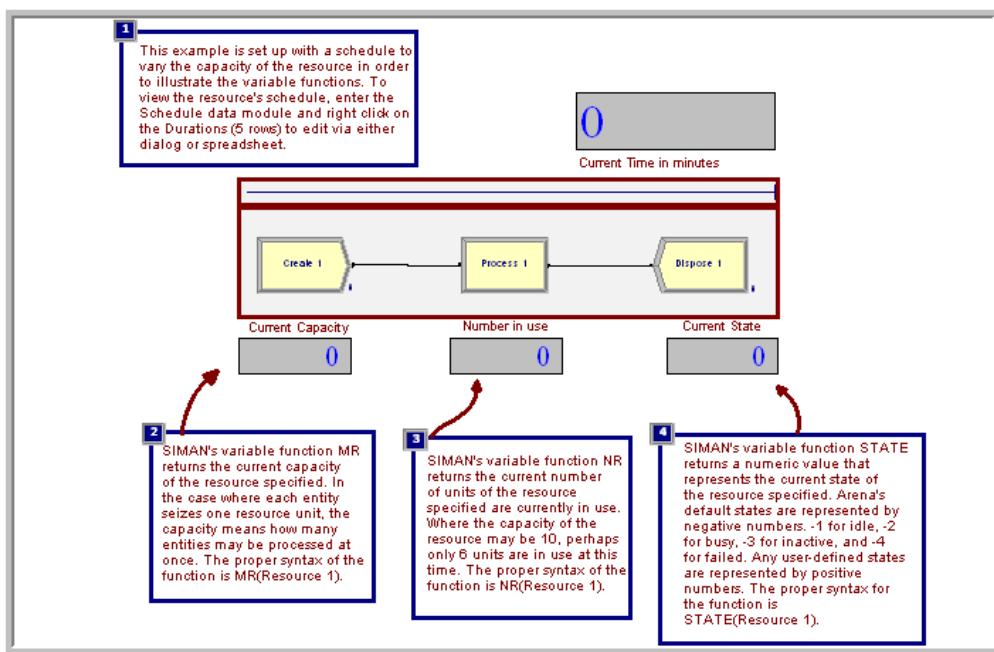


Figure 6.8.: Scheduled capacity change model

Resource		
Usage		
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	6.1154	(Insufficient)
Scheduled Utilization	Value	
Resource 1	0.6843	
Total Number Seized	Value	
Resource 1	1007.00	

Figure 6.9.: Scheduled capacity change model results

6. Modeling Systems with Advanced Process Concepts

Now, we can define the instantaneous utilization at time t as:

$$IU(t) = \begin{cases} 0 & NR(t) = 0 \\ 1 & NR(t) \geq MR(t) \\ NR(t)/MR(t) & \text{otherwise} \end{cases}$$

Thus, the time average instantaneous utilization is:

$$\bar{IU} = \frac{1}{T} \int_0^T IU(t) dt$$

The scheduled utilization is the time average number of busy resources divided by the time average number scheduled. As can be seen in Equation (6.1), this is the same as the total time spent busy divided by the total time available for all resource units.

$$\bar{SU} = \frac{\bar{NR}}{\bar{MR}} = \frac{\frac{1}{T} \int_0^T NR(t) dt}{\frac{1}{T} \int_0^T MR(t) dt} = \frac{\int_0^T NR(t) dt}{\int_0^T MR(t) dt} \quad (6.1)$$

If $MR(t)$ is constant, then $\bar{IU} = \bar{SU}$. The function, RESUTIL(Resource ID) returns $IU(t)$. Caution should be used in interpreting \bar{IU} when $MR(t)$ varies with time.

Now let's return to the example of the professor holding office hours. Let's suppose that the ignore option is used and consider case 2 and the 45 minute question. Let's also assume for simplicity that the professor had a take home exam 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 6.4 illustrates the calculations.

Table 6.4.: Example Calculation for professor utilization

Time interval	Busy time	Scheduled time	$NR(t)$	$MR(t)$	$IU(t)$
8 am - noon	240	240	1.0	1.0	1.0
12 - 12:30 pm	30	0	1.0	0	1.0
12:30 - 4 pm	210	210	1.0	1.0	1.0
	480	450			

From Table 6.4 we can compute \bar{NR} and \bar{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 6.4 indicates that $\overline{IU} = 1.0$ (or 100%) and $\overline{SU} = 1.06$ (or 106%). Who says that professors don't 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 6.10. The interested reader should search under, *resource statistics: Instantaneous Utilization Vs. Scheduled utilization*.

6. Modeling Systems with Advanced Process Concepts

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%.																																																																																																																							
<table border="1"> <thead> <tr> <th></th><th></th><th colspan="2">Case A</th><th colspan="2">Case B</th><th colspan="2">Case C</th><th colspan="2">Case D</th></tr> <tr> <th>Scenario</th><th>4 Hour Shift</th><th>1</th><th>2</th><th>1</th><th>2</th><th>1</th><th>2</th><th>1</th><th>2</th></tr> </thead> <tbody> <tr> <td># Busy</td><td></td><td>50</td><td>50</td><td>0</td><td>100</td><td>50</td><td>50</td><td>0</td><td>100</td></tr> <tr> <td># Scheduled</td><td></td><td>150</td><td>150</td><td>200</td><td>100</td><td>200</td><td>100</td><td>0</td><td>300</td></tr> <tr> <td>Utilization</td><td></td><td>33.33%</td><td>33.33%</td><td>0%</td><td>100%</td><td>25%</td><td>50%</td><td>0%</td><td>33.33%</td></tr> <tr> <td>Total Hours Busy</td><td></td><td colspan="2">$(50*8) = 400$</td><td colspan="2">$((0*4)+(100*4)) = 400$</td><td colspan="2">$(50*8) = 400$</td><td colspan="2">$(100*4) = 400$</td></tr> <tr> <td>Total Hours Scheduled</td><td></td><td colspan="2">$(150*8) = 1200$</td><td colspan="2">$((200*4)+(100*4)) = 1200$</td><td colspan="2">$((200*4)+(100*4)) = 1200$</td><td colspan="2">$(300*4) = 1200$</td></tr> <tr> <td>Average Number Busy</td><td></td><td colspan="2">$(50*8)/8 = 50$</td><td colspan="2">$((0*4)+(100*4))/8 = 50$</td><td colspan="2">$(50*8)/8 = 50$</td><td colspan="2">$((0*4)+(100*4))/8 = 50$</td></tr> <tr> <td>Average Number Scheduled</td><td></td><td colspan="2">$(150*8)/8 = 150$</td><td colspan="2">$((200*4)+(100*4))/8 = 150$</td><td colspan="2">$(100*4)/8 = 150$</td><td colspan="2">$((0*4)+(300*4))/8 = 150$</td></tr> <tr> <td>Scheduled Utilization</td><td></td><td colspan="2">$50/150 = 33.33\%$</td><td colspan="2">$50/150 = 33.33\%$</td><td colspan="2">$50/150 = 33.33\%$</td><td colspan="2">$50/150 = 33.33\%$</td></tr> <tr> <td>Instantaneous Utilization</td><td></td><td colspan="2">$(33.33\%*8)/8 = 33.33\%$</td><td colspan="2">$((0\%*4)+(100\%*4))/8 = 50\%$</td><td colspan="2">$((25\%*4)+(50\%*4))/8 = 37.5\%$</td><td colspan="2">$((0\%*4)+(33.33\%*4))/8 = 16.67\%$</td></tr> </tbody> </table> <th data-kind="ghost"></th>			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		$(150*8)/8 = 150$		$((200*4)+(100*4))/8 = 150$		$(100*4)/8 = 150$		$((0*4)+(300*4))/8 = 150$		Scheduled Utilization		$50/150 = 33.33\%$		$50/150 = 33.33\%$		$50/150 = 33.33\%$		$50/150 = 33.33\%$		Instantaneous Utilization		$(33.33\%*8)/8 = 33.33\%$		$((0\%*4)+(100\%*4))/8 = 50\%$		$((25\%*4)+(50\%*4))/8 = 37.5\%$		$((0\%*4)+(33.33\%*4))/8 = 16.67\%$										
		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		$(150*8)/8 = 150$		$((200*4)+(100*4))/8 = 150$		$(100*4)/8 = 150$		$((0*4)+(300*4))/8 = 150$																																																																																																															
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 6.10.: Utilization example from Help files

6.2.3. Resource Failure Modeling

As mentioned, a resource has 4 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 3 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 6.11. Notice the similarity to scheduled capacity changes. You can think of this situation as a failure "clock" ticking. The up-time 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.

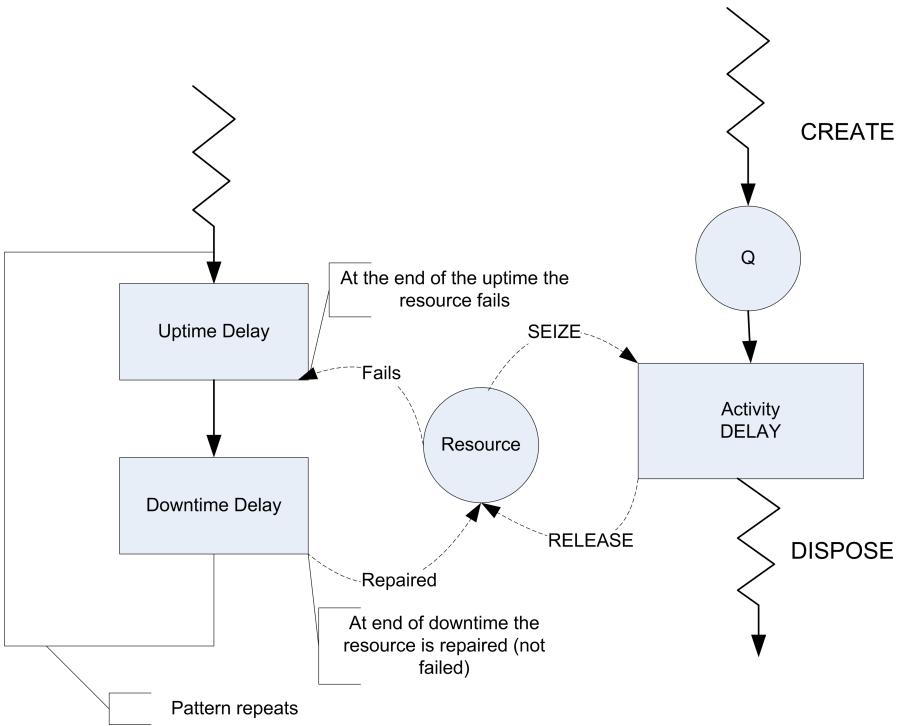


Figure 6.11.: Activity diagram for failure modeling

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.

Figure 6.12 and Figure 6.13 illustrate how to define count based and time based failures within the FAILURES module.

In both cases, the up-time, downtime, or count fields can all be expressions. In the case of the time based failure (Figure 6.13) the field *up-time 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 re-scheduled and the time to failure starts again. The up-time 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 fashion, the resource only accumulates time until failure during its busy time. Figure 6.14 and Figure 6.15 illustrate how to attach the failures to resources.

Notice that in the case of Figure 6.15}, you can clearly see that a resource can have multiple fail-

6. Modeling Systems with Advanced Process Concepts

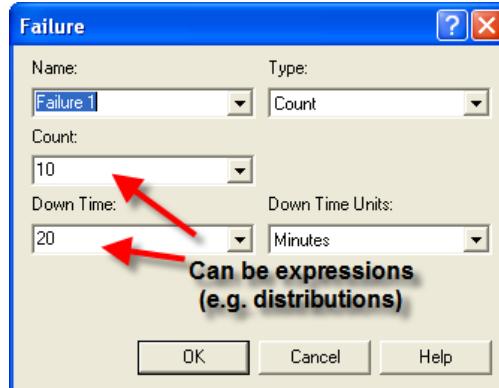


Figure 6.12.: Count cased failure dialog

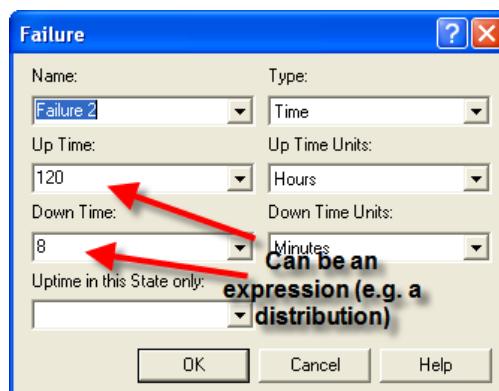


Figure 6.13.: Time based failure dialog

ures. 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 failed state. You are encouraged to run the model. You might also look at the *Smarts123.doe* file for other ideas about displaying failures.

The next section illustrates how to use resource scheduled capacity changes to effectively model a system with non-stationary arrivals.

6.3. Job Fair Example with Non-Stationary Arrivals

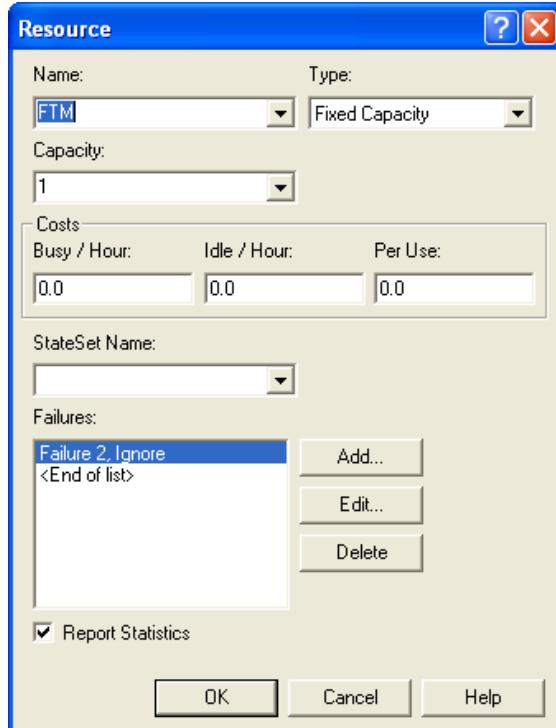


Figure 6.14.: Resource with failure attached

Failures									
	Failure Name	Failure Rule	Each resource can have multiple failures. The same failure can be used on other resources.						
	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures	Report Statistics
1	FTM	Fixed Capacity	1	0.0	0.0	0.0		1 rows	<input checked="" type="checkbox"/>
2	FCNT	Fixed Capacity	1	0.0	0.0	0.0		1 rows	<input checked="" type="checkbox"/>

Double-click here to add a new row.

Figure 6.15.: Resource with failures data sheet view

6.3. Job Fair Example with Non-Stationary Arrivals

In Chapter 3, we modeled a STEM Career Fair Mixer and illustrated some of the basic modeling techniques associated with process modeling. For example, we saw how to generate entities (students) and have them experience simple processes before leaving the system. In Chapter 4, we embellished the STEM Mixer model by more realistically modeling the closing of the mixer and illustrated how to route entities within a system. In this section, we return to the STEM Career Fair Mixer model in order to illustrate the modeling of non-stationary arrivals and the collection of statistics for such situations. We will first update the system description to reflect the new situation and then discuss the building and analysis of the new model. In this new

6. Modeling Systems with Advanced Process Concepts

modeling situation, we will see how to utilize the SCHEDULE module for the case of varying the mean arrival rate and varying the resource capacity.

Example 6.1 (Non-Stationary STEM Fair Mixer). Suppose new data on the arrival behavior of the students have become available. Over the course of the 6 hours that the mixer is open, the new data indicates that, while on average, the arrival rate is still 30 per hour, the rate varies significantly by the time of day. Let's assume that the Mixer opens at 2 pm and lasts until 8 pm. Data was collected over 30 minute intervals during the 6 hours of the mixer and it showed that there was a larger arrival rate during the middle hours of the operating hours than during the beginning or ending hours of the data. The Table 6.5 summarizes the arrival rate for each 30 minute period throughout the 6 hour period.

Table 6.5.: Student hourly arrival rates by 30 minute periods

Period	Time Frame	Duration	Mean Arrival Rate per Hour
1	2 - 2:30 pm	30	5
2	2:30 – 3 pm	30	10
3	3 - 3:30 pm	30	15
4	3:30 – 4 pm	30	25
5	4 - 4:30 pm	30	40
6	4:30 – 5 pm	30	50
7	5 - 5:30 pm	30	55
8	5:30 – 6 pm	30	60
9	6 - 6:30 pm	30	60
10	6:30 – 7 pm	30	30
11	7 - 7:30 pm	30	5
12	7:30 – 8 pm	30	5

Because of the time varying nature of the arrival of students, the STEM mixer organizers have noticed that during the peak period of the arrivals there are substantially longer lines of students waiting to speak to a recruiter and often there are not enough tables within the conversation area for students. Thus, the mixer organizers would like to advise the recruiters as to how to staff their recruiting stations during the event. As part of the analysis, they want to ensure that 90 percent of the students experience an average system time of 35 minutes or less during the 4:30-6:30 pm time frame. They would also like to measure the average number of students waiting for recruiters during this time frame as well as the average number of busy recruiters at the MalWart and JHBunt recruiting stations. You should first perform an analysis based on the current staffing recommendations from the previous modeling and then compare those results to a staffing plan that allows the recruiters to change their assigned number of recruiters by hour.

Everything else about the operation of the system remains the same as described in Chapter 4.

The solution to this modeling situation will put into practice the concepts introduced in the last two sections involving arrival schedules and resource schedules. That aspect of the modeling will turn out to be relatively straightforward. However, the requirement to collect statistics over the peak attendance period from 4:30 – 6:30 pm will require some new concepts on how to collect statistics by time periods.

6.3.1. Collecting Statistics by Time Periods

For some modeling situations, we need to be able to collect statistics during specific intervals of time. To be able to do this, we need to be able to observe what happens during those intervals. In general, to collect statistics during a particular interval of time will require the scheduling of events to observe the system (and its statistics) at the start of the period and then again at the end of the period in order to record what happened during the period. Let's define some variables to clarify the concepts. We will start with a discussion of collecting statistics for tally-based data.

Let $(x_1, x_2, x_3, \dots, x_{n(t)})$ be a sequence of observations up to and including time t .

Let $n(t)$ be the number of observations up to and including time t .

Let $s(t)$ be the cumulative sum of the observations up to and including time t . That is,

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

Let $\bar{x}(t)$ be the cumulative average of the observations up to and including time t . That is,

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

We should note that we have $s(t) = \bar{x}(t) \times n(t)$. Let t_b be the time at the beginning of a period (interval) of interest and let t_e be the time at the end of a period (interval) of interest such that $t_b \leq t_e$. Define $s(t_b, t_e]$ as the sum of the observations during the interval, $(t_b, t_e]$. Clearly, we have that,

$$s(t_b, t_e] = s(t_e) - s(t_b)$$

Define $n(t_b, t_e]$ as the count of the observations during the interval, $(t_b, t_e]$. Clearly, we have that,

6. Modeling Systems with Advanced Process Concepts

$$n(t_b, t_e] = n(t_e) - n(t_b)$$

Finally, we have that the average during the interval, $(t_b, t_e]$ as

$$\bar{x}(t_b, t_e] = \frac{s(t_b, t_e]}{n(t_b, t_e]}$$

Thus, if during the simulation we can observe, $s(t_b, t_e]$ and $n(t_b, t_e]$, we can compute the average during a specific time interval. To do this in a simulation, we can schedule an event for time, t_b , and observe, $s(t_b)$ and $n(t_b)$. Then, we can schedule an event for time, t_e , and observe, $s(t_e)$ and $n(t_e)$. Once these values are captured, we can compute $\bar{x}(t_b, t_e]$. If we observe this quantity across multiple replications, we will have the average that occurs during the defined period.

So far, the discussion has been about tally-based data. The process is essentially the same for time-persistent data. Recall that a time-persistent variable typically takes on the form of a step function. For example, let $y(t)$ represents the value of some state variable at any time t . Here $y(t)$ will take on constant values during intervals of time corresponding to when the state variable changes, for example. $y(t) = \{0, 1, 2, 3, \dots\}$. $y(t)$ is a curve (a step function in this particular case) and we compute the time average over the interval $(t_b, t_e]$. as follows.

$$\bar{y}(t_b, t_e] = \frac{\int_{t_b}^{t_e} y(t) dt}{t_e - t_b}$$

Similar to the tally-based case, we can define the following notation. Let $a(t)$ be the cumulative area under the state variable curve.

$$a(t) = \int_0^t y(t) dt$$

Define $\bar{y}(t)$ as the cumulative average up to and including time t , such that:

$$\bar{y}(t) = \frac{\int_0^t y(t) dt}{t} = \frac{a(t)}{t}$$

Thus, $a(t) = t \times \bar{y}(t)$. So, if we have a function to compute $\bar{y}(t)$ we have the ability to compute,

$$\bar{y}(t_b, t_e] = \frac{\int_{t_b}^{t_e} y(t) dt}{t_e - t_b} = \frac{a(t_e) - a(t_b)}{t_e - t_b}$$

Again, a strategy of scheduling events for t_e and t_e can allow you to observe the required quantities at the beginning and end of the period and then observe the desired average.

6.3. Job Fair Example with Non-Stationary Arrivals

The key to collecting the statistics on observation-based over an interval of time within Arena is the usage of the `TAVG(tally ID)` and the `TNUM(tally ID)` functions within Arena. The tally ID is the name of the corresponding tally variable, typically defined within the RECORD module or when defining a Tally set. Using the previously defined notation, we can note the following: $TAVG(tally ID) = \bar{x}(t)$ and $TNUM(tally ID) = n(t)$. Thus, we can compute $s(t)$ for some tally variable via:

$$s(t) = TAVG(tally ID) * TNUM(tally ID)$$

Given that we have the ability to observe $n(t)$ and $s(t)$ via these functions, we have the ability to collect the period statistics for any interval.

To make this more concrete, let's collect statistics for every hour of the day for an 8-hour simulation. In what follows, we will outline via pseudo-code how to implement this situation. Suppose we are interested in collecting the average time customers spend in a queue for a simple single server queueing system (e.g. a M/M/1 queue) for each hour of an 8 hour day. Let's call the queue, `ServerQ` and define a tally variable called `qTimeTally` that collects the average time in the queue.

The first step is to define a clock entity that can keep track of each hour of the day. We will use the clock entity to schedule the events and record the desired statistics for each period. The clock entity will loop after delaying for the period length and increment the period indicator. At the end of the period, we will tabulate $n(t)$ and $s(t)$ using `TAVG()` and `TNUM()`, compute the average during the period and then prepare for the next hour by saving the cumulative statistics.

First, we define the variables, attributes and sets that we are going to use.

VARIABLES:

```
// to keep track of the current hour, the 0th hour is the 1st period
vPeriod
```

Attributes:

```
mySum// the cumulative sum at the beginning of the period
myNum // the cumulative number at the beginning of the period
myPeriodSum // the sum during the period
myPeriodNum // the count during the period
myPeriodAvg // the average computed within the period
```

Sets:

```
WaitTimeTallySet as a tally set with 8 elements, 1 for each period
(hour) of the day
```

Then, the pseudo-code for the clock-entity operation is as follows:

```
CREATE 1 entity at time 0.0
A: ASSIGN vPeriod = vPeriod + 1
```

6. Modeling Systems with Advanced Process Concepts

```
mySum = 0.0
myNum = 0
DELAY for 1 hour
BEGIN ASSIGN
// collect the cumulative sum and count
myPeriodSum = TAVG(QTimeTally)*TNUM(QTimeTally) - mySum
myPeriodNum = TNUM(QTimeTally) - myNum
// save the current sum again for next period
mySum = TAVG(QTimeTally)*TNUM(QTimeTally)
myNum = TNUM(QTimeTally)
END ASSIGN
DECIDE IF myPeriodNum > 0
RECORD (myPeriodSum/myPeriodNum) by Set: WaitTimeTallySet using index, vPeriod
END DECIDE
Go To: A
```

If there are multiple performance measures that need collecting, then this kind of logic can be repeated. In addition, by using sets, arrays, or arrayed expressions, you could define many variables and loop through all that are necessary to compute many performance measures for the periods. This can become quite tedious for a large number of performance measures. Also, note that the above logic assumes that we are simulating the system of interest for 8 hours, such that the replication length is set at 8 hours and the simulation will stop at that time. If you are collecting statistics for periods of time that do not span the entire simulation run length, then you need to provide additional logic for the clock-entity so that you ensure valid collection intervals.

Finally, to compute the time-persistent averages within Arena we can proceed in a very similar fashion as previously described by using the `DAVG(Dstat ID)` function. The `DAVG(Dstat ID)` function is $\bar{y}(t)$ the cumulative average of a time-persistent variable. Thus, we can easily compute $a(t) = t \times \bar{y}(t)$ via `DAVG(Dstat ID)*TNOW`. Therefore, we can use the same strategy as noted in the illustrative pseudo-code to compute the time average over a given period of time, $\bar{y}(t_b, t_e)$.

Fortunately, Arena has a method to automate this approach within the STATISTIC module on the Advanced Process Panel. Within the module you can define a time-persistent statistic and then provide a pattern for collecting the average over user defined periods. The Smarts file, "Statistics Capturing Variable Metric over a specified Time Period.doe" illustrates the use of this option. In the dialog shown here, the variable, `vMoney_in_the_Bank`, is observed starting at time 3 hours after the simulation starts, with an observation period of 3 hours. In addition, the observation is repeated every 8 hours. That is, after an 8-hour period, we start the observation period of 3 hours again. For additional details, the interested reader should refer to the Arena Help on these options, which will provide more of the technical details of the meaning of the various options. We will illustrate the use of this functionality when developing the updated STEM Mixer model.

6.3. Job Fair Example with Non-Stationary Arrivals

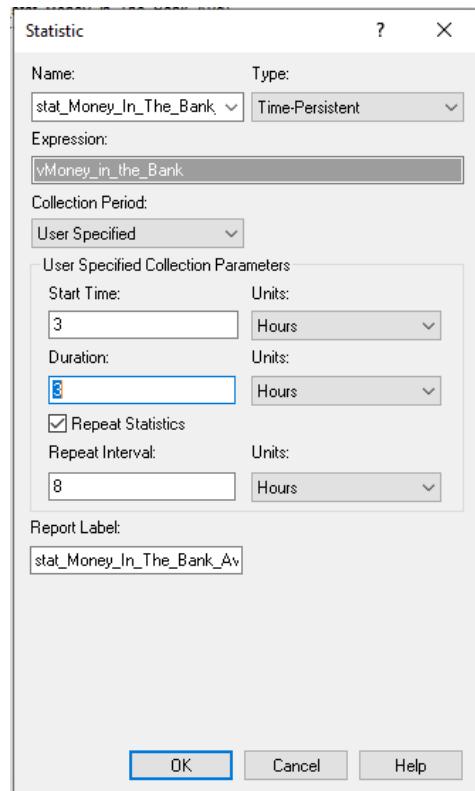


Figure 6.16.: Capturing variable metric over a specified time period

6.3.2. Modeling the Statistical Collection

Now that we have a better understanding of the statistical collection requirements for the revised STEM mixer situation, we can build the model. First, we will outline the collection of the statistics during the peak period using the concepts from the last section. Then, we will implement the model using Arena.

The pseudo-code for this situation is actually simpler than illustrated in the last section because we only have to collect statistics during the peak period and it occurs only once during the 6 hour operating horizon. In what follows, we will collect both the system time and the average number of students waiting for a recruiter during the peak period. We will also see how to collect the average number waiting during the peak period using Arena's STATISTIC module.

Define the following additional variables, attributes, and statistics to add to the existing STEM Mixer model of Chapter 4.

VARIABLES:

```
// time that the period starts, e. g. 4:30  
vTimeOfPeriodStart = 150 minutes
```

6. Modeling Systems with Advanced Process Concepts

```
// the length of the peak period
vPeakPeriodLength = 120 minutes

ATTRIBUTES:
// cumulative sum for system time at the beginning of the peak period
myCumSumSysTime
// cumulative number of system time observations at the beginning of the peak period
myCumNumSysTime
// cumulative area of the number of students waiting at the beginning of the peak period
myCumAreaWaitingStudents

TALLIES:
Tally System Time

TIME PERSISTENT Statistics:
NumStudentsWaitingTP, NQ(MalMart) + NQ(JHBunt)
```

Now, we can outline the modeling to collect the statistics by period.

```
CREATE 1 entity at time vTimeOfPeriodStart
BEGIN ASSIGN
    myCumSumSysTime = TAVG(Tally System Time)*TNUM(Tally System Time)
    myCumNumSysTime = TNUM(Tally System Time)
    myCumAreaWaitingStudents = DSTAT(NumStudentsWaitingTP)*TNOW
END ASSIGN
DELAY vPeakPeriodLength minutes
BEGIN ASSIGN
    myPeriodSum = TAVG(Tally System Time)*TNUM(Tally System Time) - myCumSumSysTime
    myPeriodNum = TNUM(Tally System Time) - myCumNumSysTime
    myPeriodArea = DSTAT(NumStudentsWaitingTP)*TNOW - myCumAreaWaitingStudents
END ASSIGN
DECIDE IF myPeriodNum > 0
    RECORD (myPeriodSum/myPeriodNum) as SystemTimeDuringPeriod
END DECIDE
RECORD (myPeriodArea/vPeakPeriodLength) as TimeAvgNumWaitingDuringPeriod
DISPOSE
```

Notice that within this pseudo-code, we only have the two events (one at the beginning and one at the end) of the delay representing the observation period in which to capture the statistics. Before the period delay, we capture the statistics at the beginning of the period. Then, after the period, we can compute what happened during the period and then record the statistics for the period. Also notice the use of variables to represent the peak period length and when the period starts. This allows these variables to be changed more easily if you decide to experiment with the parameter settings.

6.3.3. Implementing the Model in Arena

Now we are ready to implement the changes to the model developed in Chapter 4 section 4.1. There are really just two major changes to make 1) implementing the non-stationary arrivals and 2) implementing the statistical collection for the peak period. To start, we need to take the mean arrival rate information found in Example 6.1 and define a schedule for the arrival of students to the mixer.

The screenshot shows the 'Schedule - Basic Process' dialog. A table lists a single row named 'StudentArrivalSchedule' with Type 'Arrival', Time Units 'Minutes', Scale Factor '1.0', and Durations '12 rows'. A modal window titled 'Durations' displays a 12x3 grid of values and durations:

	Value	Duration
1	5	30
2	10	30
3	15	30
4	25	30
5	40	30
6	50	30
7	55	30
8	60	30
9	60	30
10	30	30
11	5	30
12	5	30

Figure 6.17.: Student arrival schedule for 30 minute intervals

Figure 6.17 shows the Arena arrival schedule data sheet view. Because the time units are set to minutes, the (value, duration) pairs are entered with the duration values as 30 minutes. Also, because the arrival rate data is already per hour, we can simply enter the mean arrival rates. If the provided mean arrival rate data was not per hour, then we would have needed to convert them to a per hour basis before entering them into the arrival schedule. Next, the arrival schedule must be connected to the CREATE module used to create the student entities.

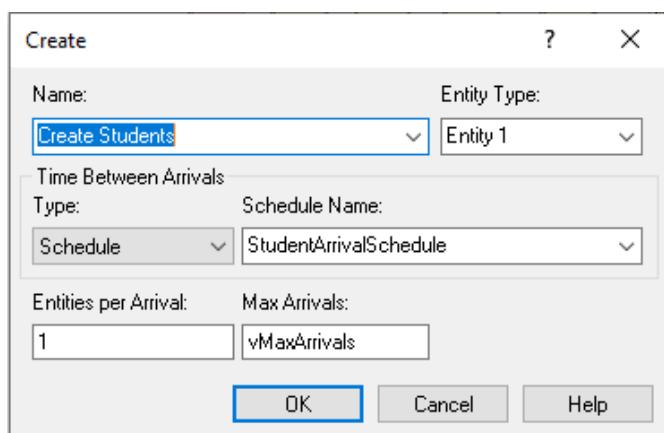


Figure 6.18.: Updated CREATE module with arrival schedule

6. Modeling Systems with Advanced Process Concepts

Figure 6.18 illustrates the updated CREATE module. Notice that the type of arrival has been designated as *Schedule* and the name of the schedule from Figure 6.17 has been supplied. As discussed in Chapter 4, we still turn off the CREATE module by setting `vMaxArrivals` to 0 by using an entity scheduled to occur at the end of the mixer. However, because we used an arrivals schedule we could also have specified one extra interval in the arrival schedule with the (rate, duration) pair as (0,). A blank duration is interpreted as infinity. Thus, rate would be set the mean arrival rate to 0 when that interval occurs and it would remain at 0 for the remainder of the simulation. This would effectively shut off the CREATE module.

Now we will implement the pseudo-code of the last section within Arena. An overview of the Arena flow chart modules can be found in Figure 6.19.

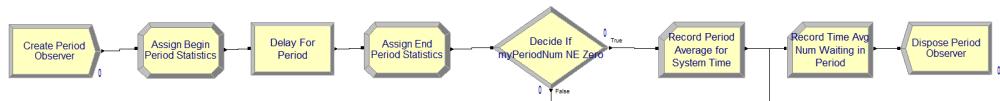


Figure 6.19.: Overview of statistical collection logic

To implement this logic, we will first update the variables, attributes, and time persistent variables. Figure 6.20 and 6.21 show the updated attributes and variables to be used in the revised model. The variables and attribute names match what was presented in the pseudo-code.

Attribute - Basic Process		Comment	Rows	Columns	Data Type	Initial Values
1	myArrivalTime	holds when students arrive at the mixer			Real	0 rows
2	myType	1= recruiters first, 2 = conversation first, 3 = timid			Real	0 rows
3	mySysTime	total time spent at the mixer			Real	0 rows
4	myNumVisits	the number of recruiters visited			Real	0 rows
5	myFirstRecruiter	the recruiter visited first			Real	0 rows
6	my45MinuteFlag	indicates that visited 1 recruiter and left because of 45 minutes in mixer			Real	0 rows
7	myCumSumSysTime	holds cum sum of systems times during peak period			Real	0 rows
8	myCumNumSysTime	holds cum num of system times observed during peak period			Real	0 rows
9	myCumAreaWaitingStudents	holds cum area of number waiting during peak period			Real	0 rows
10	myPeriodSum	the sum of the system times during the peak period			Real	0 rows
11	myPeriodNum	the num of system times observed during the peak period			Real	0 rows
12	myPeriodArea	the area associated with the number waiting during the peak period			Real	0 rows

Figure 6.20.: Updated attributes

Figure 6.22 illustrates the updated definition of additional time persistent variables to collect statistics during various time periods. The two definitions on rows 2 and 3 collect the same performance measure, the average number of students waiting for either recruiter during the peak period. The definition on row 2 is used within the implementation of the previously discussed pseudo-code. The definition on row 3 will use Arena's enhance statistical collect by period functionality to start the collection at time 150 and last for 120 minutes. We will see that the results of both methods for collecting this performance measure will match exactly.

Then, there are additional time-persistent performance measures defined for the peak period and for the collection of hourly statistics. Rows 4 and 5 of Figure 6.22 collect the utilization of

6.3. Job Fair Example with Non-Stationary Arrivals

Variable - Basic Process									
	Name	Comment	Rows	Columns	Data Type	Clear Option	File Name	Initial Values	Report Statistics
1	vNumInSys	tracks number of students in the system			Real	System		0 rows	<input checked="" type="checkbox"/>
2	vClosed	indicates if mixer is closing			Real	System		0 rows	<input type="checkbox"/>
3	vNumAtJHBunt	tracks number of students visiting JHBunt recruiters			Real	System		0 rows	<input type="checkbox"/>
4	vNumAtMalMart	tracks number of students visiting MalMart recruiters			Real	System		0 rows	<input type="checkbox"/>
5	vWaningTime	in minutes			Real	System		1 rows	<input type="checkbox"/>
6	vMixerTimeLength	in minutes			Real	System		1 rows	<input type="checkbox"/>
7	vMaxArrivals	used to shut off student arrivals			Real	System		1 rows	<input type="checkbox"/>
8	vPeakPeriodLength	length of peak period in minutes			Real	System		1 rows	<input type="checkbox"/>
9 ►	vTimeOfPeriodStart	time that the peak period starts in minutes			Real	System		1 rows	<input type="checkbox"/>

Figure 6.21.: Updated variables

Time Persistent - Statistics									
	Name/Report Label	Expression	Collection Period	Start Time	Units	Duration	Units	Repeat Statistics	Output File
1	Avg Num In System	vNumInSys	Entire Replication	0.0	Hours	0.0	Hours	<input checked="" type="checkbox"/>	
2	NumStudentsWaitingTP	NQ(Visit JHBunt.Queue) + NQ(Visit MalWart.Queue)	Entire Replication	0.0	Hours	0.0	Hours	<input checked="" type="checkbox"/>	
3	NumStudentsWaitingTP2	NQ(Visit JHBunt.Queue) + NQ(Visit MalWart.Queue)	User Specified	150	Minutes	120	Minutes	<input type="checkbox"/>	
4	UtilMalWartDuringPeriod	ResUtil(rMalWartRecruiters)	User Specified	150	Minutes	120	Minutes	<input type="checkbox"/>	
5	UtilJHBuntDuringPeriod	ResUtil(rJHBuntRecruiters)	User Specified	150	Minutes	120	Minutes	<input type="checkbox"/>	
6	NumBusyHour1JHBunt	NR(rJHBuntRecruiters)	User Specified	0.0	Hours	1.0	Hours	<input type="checkbox"/>	
7	NumBusyHour2JHBunt	NR(rJHBuntRecruiters)	User Specified	1.0	Hours	1.0	Hours	<input type="checkbox"/>	
8	NumBusyHour3JHBunt	NR(rJHBuntRecruiters)	User Specified	2.0	Hours	1.0	Hours	<input type="checkbox"/>	
9	NumBusyHour4JHBunt	NR(rJHBuntRecruiters)	User Specified	3.0	Hours	1.0	Hours	<input type="checkbox"/>	
10	NumBusyHour5JHBunt	NR(rJHBuntRecruiters)	User Specified	4.0	Hours	1.0	Hours	<input type="checkbox"/>	
11	NumBusyHour6JHBunt	NR(rJHBuntRecruiters)	User Specified	5.0	Hours	1.0	Hours	<input type="checkbox"/>	
12	NumBusyHour1MalWart	NR(rJHBuntRecruiters)	User Specified	0.0	Hours	1.0	Hours	<input type="checkbox"/>	
13	NumBusyHour2MalWart	NR(rJHBuntRecruiters)	User Specified	1.0	Hours	1.0	Hours	<input type="checkbox"/>	
14	NumBusyHour3MalWart	NR(rJHBuntRecruiters)	User Specified	2.0	Hours	1.0	Hours	<input type="checkbox"/>	
15 ►	NumBusyHour4MalWart	NR(rJHBuntRecruiters)	User Specified	3.0	Hours	1.0	Hours	<input type="checkbox"/>	
16	NumBusyHour5MalWart	NR(rJHBuntRecruiters)	User Specified	4.0	Hours	1.0	Hours	<input type="checkbox"/>	
17	NumBusyHour6MalWart	NR(rJHBuntRecruiters)	User Specified	5.0	Hours	1.0	Hours	<input type="checkbox"/>	

Figure 6.22.: Updated time persistent statistic definitions

the resources used to model the JHBunt and MalMart recruiters using the `ResUtil()` function. This will allow us to note the utilization during the peak period. Then, rows 6 through 17 simply collect the average number of busy units of resource for JHBunt and MalMart for each hour of operation of the mixer. These statistics will facilitate determining how many recruiters to staff during each hour of operation.

To implement the modules found in Figure 6.19, we need to use CREATE, ASSIGN, DELAY, DECIDE, and RECORD modules that follow the previously described pseudo-code. Figure 6.23 shows the creating of the logical entity to collect statistics for the peak period.

Figure 6.24 shows the ASSIGN modules used to capture the statistics at the beginning and end of the period. Notice the use of the `TAVG()`, `TNUM()`, and `DAVG()` functions as discussed in the pseudo-code.

Figure 6.25 illustrates the use of a DECIDE module to check if there were no observations during the peak period in order to avoid a divide by zero error when computing the system time for the peak period.

Figures 6.26 and 6.27 present the RECORD modules for capturing the peak period system time and average number of students waiting for either recruiter.

6. Modeling Systems with Advanced Process Concepts

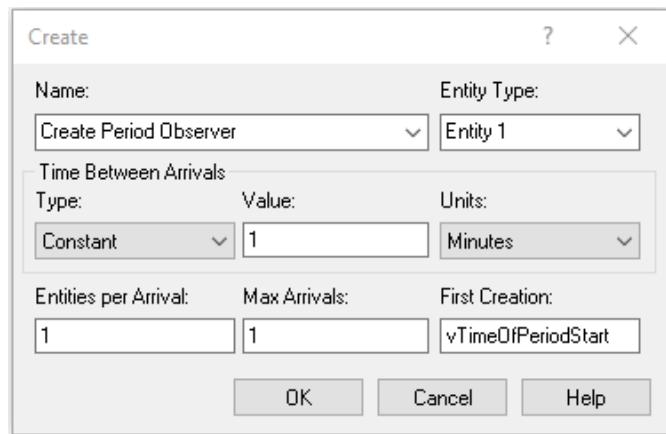


Figure 6.23.: Creating the period observer

Assignments			
	Type	Attribute Name	New Value
1	Attribute	myCumSumSysTime	TAVG(Tally System Time) * TNUM(Tally System Time)
2	Attribute	myCumNumSysTime	TNUM(Tally System Time)
3	Attribute	myCumAreaWaitingStudents	DAVG(NumStudentsWaitingTP) * TNOW

Assignments			
	Type	Attribute Name	New Value
1	Attribute	myPeriodSum	TAVG(Tally System Time) * TNUM(Tally System Time) - myCumSumSysTime
2	Attribute	myPeriodNum	TNUM(Tally System Time) - myCumNumSysTime
3	Attribute	myPeriodArea	DAVG(NumStudentsWaitingTP) * TNOW - myCumAreaWaitingStudents

Figure 6.24.: Assign logic to capture period Statistics

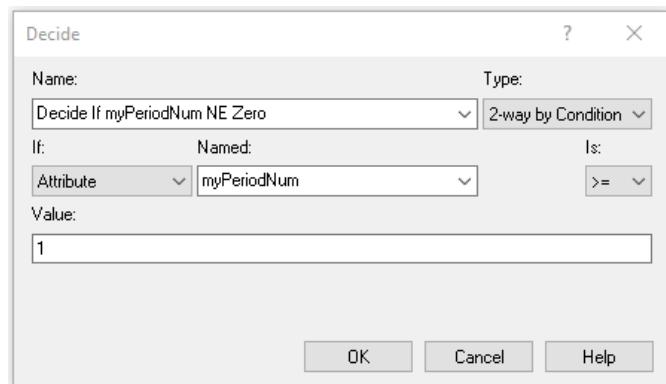


Figure 6.25.: Checking for zero in the denominator

6.3. Job Fair Example with Non-Stationary Arrivals

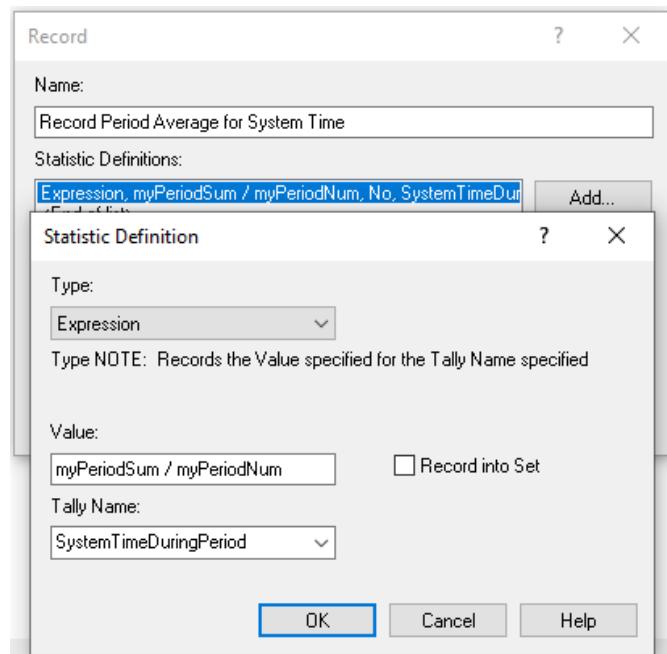


Figure 6.26.: Updated time persistent statistic definitions

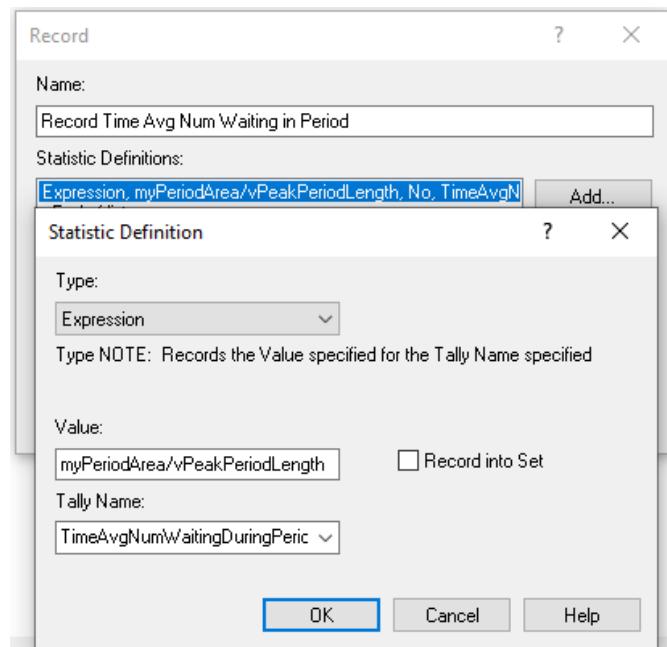


Figure 6.27.: Updated time persistent statistic definitions

6. Modeling Systems with Advanced Process Concepts

The entire Arena model is found in the file *STEM_MixerArrivalsScheduleOnly.doe*. This implementation has not yet considered the resource schedule constructs that were previously discussed. We will run this model and review the statistics to understand how to set up the resource capacity change. The base case scenario that we will perform is to run the model using the previously specified capacity of 3 units for JHBunt and 2 units for MalMart. We will see how the non-stationary arrivals affect the performance of the system.

Tally

Expression	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Num at JHBunt at Closing	18.1000	2.71	1.0000	31.0000	1.0000	31.0000
Num at MalMart at Closing	3.4333	1.75	0.00	16.0000	0.00	16.0000
Num Conversing at Closing	0.7333	0.31	0.00	3.0000	0.00	3.0000
Num in Sys at Closing	22.5333	3.04	7.0000	49.0000	7.0000	49.0000
SystemTimeDuringPeriod	42.2770	2.28	30.9415	55.5056	30.9415	55.5056
Tally Prob Leave 45 mintues	0.1984	0.04	0.00	0.3707	0.00	1.0000
TimeAvgNumWaitingDuringPeriod	25.9556	2.34	14.6053	38.1056	14.6053	38.1056
Type 1 Prob LT 30 min	0.3435	0.03	0.1667	0.4679	0.00	1.0000
Interval	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Tally System Time	51.8295	2.41	39.5457	67.5708	1.4472	152.48

Figure 6.28.: Base scenario tally results for non-stationary arrivals and no resource capacity change

Time Persistent	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Avg Num In System	23.1771	1.35	17.5694	30.2989	0.00	90.0000
NumBusyHour1JHBunt	0.4568	0.07	0.1706	0.9046	0.00	3.0000
NumBusyHour1MalWart	0.4568	0.07	0.1706	0.9046	0.00	3.0000
NumBusyHour2JHBunt	1.3076	0.13	0.7590	1.9503	0.00	3.0000
NumBusyHour2MalWart	1.3076	0.13	0.7590	1.9503	0.00	3.0000
NumBusyHour3JHBunt	2.6828	0.09	2.0738	3.0000	0.00	3.0000
NumBusyHour3MalWart	2.6828	0.09	2.0738	3.0000	0.00	3.0000
NumBusyHour4JHBunt	3.0000	0.00	3.0000	3.0000	3.0000	3.0000
NumBusyHour4MalWart	3.0000	0.00	3.0000	3.0000	3.0000	3.0000
NumBusyHour5JHBunt	3.0000	0.00	3.0000	3.0000	3.0000	3.0000
NumBusyHour5MalWart	3.0000	0.00	3.0000	3.0000	3.0000	3.0000
NumBusyHour6JHBunt	2.9361	0.08	1.9556	3.0000	0.00	3.0000
NumBusyHour6MalWart	2.9361	0.08	1.9556	3.0000	0.00	3.0000
NumStudentsWaitingTP	15.8614	1.26	10.3000	22.7775	0.00	78.0000
NumStudentsWaitingTP2	25.9556	2.34	14.6053	38.1056	0.00	70.0000
UtilJHBuntDuringPeriod	0.9928	0.01	0.9341	1.0000	0.00	1.0000
UtilMalWartDuringPeriod	0.9746	0.01	0.8919	1.0000	0.00	1.0000

Figure 6.29.: Base scenario time persistent results for non-stationary arrivals and no resource capacity change

Examining Figure 6.28, the first thing to note is that the overall system time for the students has increased to about 52 minutes versus the 27.6 minutes in Chapter 4. Clearly, the non-stationary

6.3. Job Fair Example with Non-Stationary Arrivals

arrival behavior of the students has a significant affect on the system and should be accounted for within the modeling. By looking at Figure 6.29, the reason for the increase in system time becomes clear. The utilization of the recruiting stations is nearly 100 percent (99% and 97%) during the peak period. Utilization this high will cause a significant line that will take some time to be worked down as the arrival rate eventually decreases. This can be better noted by reviewing the average number of busy units by hour of operation within Figure 6.29. Clearly, hours 4, 5, and 6 all have the reported average number busy very near or at the capacity limit of the resources. This indicates that we should increase the capacity during those time periods. Finally, please note the value of the statistic, `TimeAvgNumWaitingDuringPeriod` on Figure 6.28 of 25.9556. In addition, note on Figure 6.29, the value of time-persistent statistic, `NumStudentsWaitingTP2`, as also 25.9556. We see that the two methods of capturing this statistic via the *by hand* method and by enhanced functionality for period based collection of time-persistent statistics resulted in exactly the same estimates. Now you can better understand what Arena is doing to collect period based statistics.

A easy way to further analyze this situation is to set the capacity of the resources to infinity and review the estimated number of busy resources by hour of the day. Then, we can specify a capacity by hour that results in a desired utilization for each hour of the day.

Time Persistent	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Avg Num In System	8.3597	0.29	6.7826	10.0678	0.00	32.0000
NumBusyHour1JHBunt	0.4562	0.07	0.1706	0.9046	0.00	4.0000
NumBusyHour1MaWart	0.4562	0.07	0.1706	0.9046	0.00	4.0000
NumBusyHour2JHBunt	1.3598	0.16	0.7568	2.6874	0.00	7.0000
NumBusyHour2MaWart	1.3598	0.16	0.7568	2.6874	0.00	7.0000
NumBusyHour3JHBunt	3.7397	0.29	2.6934	5.2275	0.00	13.0000
NumBusyHour3MaWart	3.7397	0.29	2.6934	5.2275	0.00	13.0000
NumBusyHour4JHBunt	5.2991	0.33	3.3287	7.0671	0.00	15.0000
NumBusyHour4MaWart	5.2991	0.33	3.3287	7.0671	0.00	15.0000
NumBusyHour5JHBunt	5.0701	0.27	3.3180	6.5527	0.00	14.0000
NumBusyHour5MaWart	5.0701	0.27	3.3180	6.5527	0.00	14.0000
NumBusyHour6JHBunt	1.1652	0.18	0.2500	2.3104	0.00	7.0000
NumBusyHour6MaWart	1.1652	0.18	0.2500	2.3104	0.00	7.0000
NumStudentsWaitingTP	0.00	0.00	0.00	0.00	0.00	0.00
NumStudentsWaitingTP2	0.00	0.00	0.00	0.00	0.00	0.00
UtilJHBuntDuringPeriod	0.00000000	0.00	0.00000000	0.00000000	0.00	0.00000001
UtilMaWartDuringPeriod	0.00000000	0.00	0.00000000	0.00000000	0.00	0.00000001

Figure 6.30.: Base scenario time persistent results for non-stationary arrivals and infinite resource capacity

Figure 6.30 shows the result of running the non-stationary arrival schedule with infinite capacity for the recruiting resources. Notice how the estimated number busy units for the resources track the arrival pattern. Now considering the JHBunt recruiting station during the fourth hour of the day, we see an average number of recruiters busy of 5.2991. Thus, if we set the resource capacity of the JHBunt recruiting station to 6 recruiters during the fourth hour of the day, we would expect to achieve a utilization of about 88%, $(5.2991/6 = 0.8832)$. With this thinking in mind we can determine the capacity for each hour of the day and the expected utilization.

6. Modeling Systems with Advanced Process Concepts

Schedule - Basic Process						
	Name	Type	Time Units	Scale Factor	File Name	Durations
1	StudentArrivalSchedule	Arrival	Minutes	1.0		12 rows
2	JHBuntRecruiterSchedule	Capacity	Minutes	1.0		6 rows
3 ►	MalMartRecruiterSchedule	Capacity	Minutes	1.0		6 rows

Durations

	Value	Duration
1	1	60
2	2	60
3	4	60
4	6	60
5	6	60
6	2	60

Double-click here to add a new row.

Figure 6.31.: Possible resource capacity schedule for STEM Mixer

Resource - Basic Process									
	Name	Type	Schedule Name	Schedule Rule	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures
1	rMalMartRecruiters	Based on Schedule	MalMartRecruiterSchedule	Wait	0.0	0.0	0.0		0 rows <input checked="" type="checkbox"/>
2	rJHBuntRecruiters	Based on Schedule	JHBuntRecruiterSchedule	Wait	0.0	0.0	0.0		0 rows <input checked="" type="checkbox"/>

Figure 6.32.: Using the resource schedules in the resource module

Figure 6.31 illustrates a possible resource capacity change schedule for the MalMart recruiting station. For simplicity the same schedule will be used for the JHBunt recruiting station. The Arena file for this change can be found in the chapter files as *STEM_MixerBothSchedules.doe*. Figure 6.32 illustrates how to connect the defined resource schedule to the resource module so that the resource uses the schedule during the simulation. Figure 6.33 and Figure 6.34 present the results when using the resource schedule defined in Figure 6.31.

Tally						
Expression	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Num at JHBunt at Closing	2.5667	1.85	0.00	19.0000	0.00	19.0000
Num at MalMart at Closing	0.2000	0.18	0.00	2.0000	0.00	2.0000
Num Conversing at Closing	0.7667	0.32	0.00	3.0000	0.00	3.0000
Num in Sys at Closing	3.6333	1.96	0.00	20.0000	0.00	20.0000
SystemTimeDuringPeriod	21.1226	1.47	16.2353	29.4840	16.2353	29.4840
Tally Prob Leave 45 mintues	0.00113903	0.00	0.00	0.00602410	0.00	1.0000
TimeAvgNumWaitingDuringPeriod	4.0430	1.27	0.5959	11.7436	0.5959	11.7436
Type 1 Prob LT 30 min	0.9372	0.03	0.6577	1.0000	0.00	1.0000
Interval	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Tally System Time	22.1783	1.92	17.7793	40.5932	1.4472	163.49

Figure 6.33.: Tally results for non-stationary arrivals and time varying resource capacity

The first thing to notice is in Figure 6.33 the substantial drop in overall system time from the near 52 minutes down to about 22 minutes. Also, from Figure 6.34, we see that the utilization of the JHBunt recruiting station is still a bit high (90%) during the peak arrival period. Thus, it may be worth exploring the addition of an additional recruiter during hour 5. We leave the further exploration of this model to the interested reader.

6.4. Modeling Balking and Reneging

Time Persistent	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Avg Num In System	9.9281	0.85	7.3958	14.7685	0.00	50.0000
NumBusyHour1JHBunt	0.3950	0.05	0.1706	0.7187	0.00	2.0000
NumBusyHour1MalWart	0.3950	0.05	0.1706	0.7187	0.00	2.0000
NumBusyHour2JHBunt	1.2720	0.15	0.5258	2.0000	0.00	4.0000
NumBusyHour2MalWart	1.2720	0.15	0.5258	2.0000	0.00	4.0000
NumBusyHour3JHBunt	3.2027	0.21	2.1202	4.0000	0.00	6.0000
NumBusyHour3MalWart	3.2027	0.21	2.1202	4.0000	0.00	6.0000
NumBusyHour4JHBunt	5.4116	0.23	4.1496	6.0000	0.00	6.0000
NumBusyHour4MalWart	5.4116	0.23	4.1496	6.0000	0.00	6.0000
NumBusyHour5JHBunt	5.3117	0.25	3.6486	6.0000	0.00	6.0000
NumBusyHour5MalWart	5.3117	0.25	3.6486	6.0000	0.00	6.0000
NumBusyHour6JHBunt	1.3042	0.20	0.3178	2.2887	0.00	6.0000
NumBusyHour6MalWart	1.3042	0.20	0.3178	2.2887	0.00	6.0000
NumStudentsWaitingTP	2.1443	0.72	0.3134	8.2159	0.00	31.0000
NumStudentsWaitingTP2	4.0430	1.27	0.5959	11.7436	0.00	27.0000
UtilJHBuntDuringPeriod	0.9032	0.03	0.7992	1.0000	0.00	1.0000
UtilMalWartDuringPeriod	0.4920	0.02	0.4096	0.6012	0.00	1.0000

Figure 6.34.: Time persistent results for non-stationary arrivals and time varying resource capacity

Non-stationary arrival patterns are a fact of life in many systems that handle people. The natural daily processes of waking up, working, etc. all contribute to processes that depend on time. This section illustrated how to model non-stationary arrival processes and illustrated some of the concepts necessary when developing staffing plans for such systems. Incorporating these modeling issues into your simulation models allows for more realistic analysis; however, this also necessitates much more complex statistical analysis of the input distributions and requires careful capture of meaningful statistics. Capturing statistics by time periods is especially important because the statistical results that do not account for the time varying nature of the performance measures may mask what may be actually happening within the model. The overall average across the entire simulation time horizon may be significantly different than what occurs during individual periods of time that correspond to non-stationary situations. A good design must reflect these variations due to time.

In the next section, we continue the exploration of advanced process modeling techniques by illustrating how to model the balking and reneging of entities within queues.

6.4. Modeling Balking and Reneging

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. You can find the completed model file for this section in the chapter files under the name, *ClinicExample.doe*.

6. Modeling Systems with Advanced Process Concepts

Example 6.2 (Walk-in health care clinic). A walk-in 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 (non-urgent 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 first-come first served 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 4 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 non-urgent (low priority) patients may depart if they have to wait longer than 15 ± 5 minutes after triage. That is, a non-urgent patient may enter the clinic and begin waiting for a doctor, but if they have to wait more than 15 ± 5 minutes (uniformly distributed) they will decide to renege and leave the clinic without getting service. The clinic would like to estimate the following:

1. the average system time of each type of patient
2. the probability that low priority patients balk
3. the probability that low priority patients renege
4. the distribution of the number of customers waiting in the doctor queue

Solution to Example 6.2

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

6.4. Modeling Balking and Reneging

fact, this information can be held in expressions as illustrated in Figure 6.35. In the DISC(0 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 6.36.

Name	Rows	Columns	Expression Values
PatientServiceTime	3		3 rows
PatientTypeDist			1 rows
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 6.35.: Expressions for walk-in clinic model

Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics
vBalkCriteria			Real	System	1 rows 10	
vSearchNumb			Real	System	0 rows	

Figure 6.36.: Walk-in clinic VARIABLE module

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 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 the following pseudo-code.

```

CREATE patients
ASSIGN myType = PatientTypeDist
DECIDE
  IF (myType == 3) and (NQ(Doctor's Q) >= vBalkCriteria)
    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
  
```

6. Modeling Systems with Advanced Process Concepts

```
SEIZE 1 unit of Doctors
DELAY for service based on patient type
RELEASE 1 unit of Doctors
RECORD patient's system time
DISPOSE
ENDIF
END DECIDE
```

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 requires additional effort. There are no magic dialog boxes or modules that directly handle reneging. The key to modeling the reneging is to remember how Arena processes entities. In particular, from your study in Chapter 2, 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 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 6.37 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 to the event scheduler. At this point, the duplicate (clone) entity begins to move.

Within the simulation, no actual simulation time has advanced. That is, the clone moves at the same simulated time because it was put in the *ready* state before becoming the *active* entity. The now active clone then enters the delay module that represents the time until reneging. This

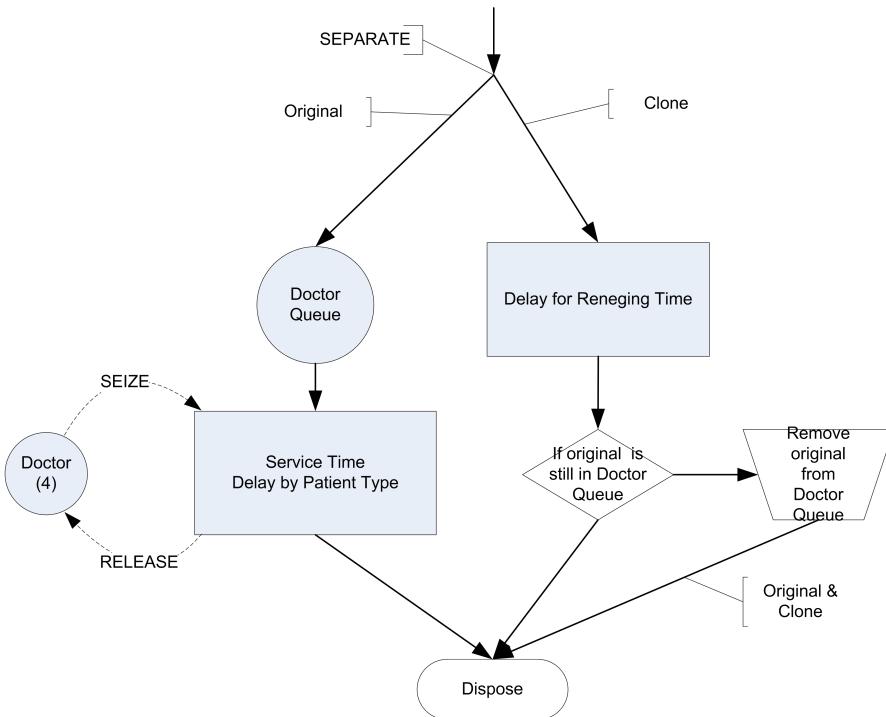


Figure 6.37.: Activity diagram for modeling reneging

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 you will need to use the SEPARATE, SEARCH, and REMOVE modules. Now, let's take a look at the entire model within .

Figure 6.38 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 6.39.

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 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 6.40. You should attempt to build this model or examine the

6. Modeling Systems with Advanced Process Concepts

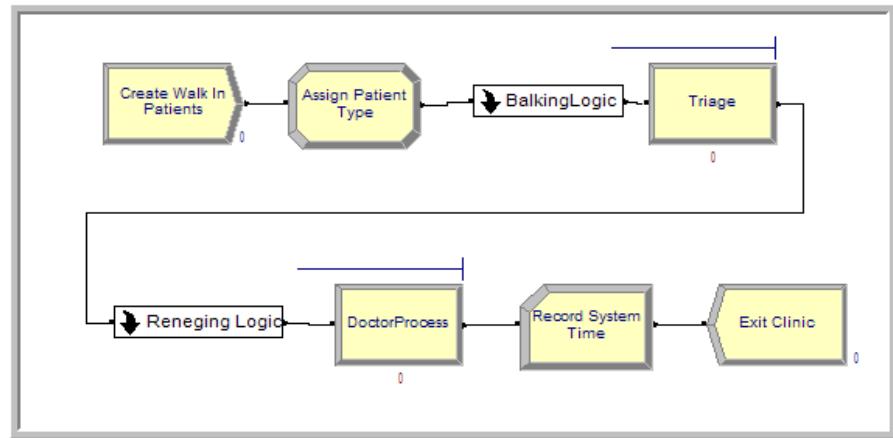


Figure 6.38.: Overview of walk-in clinic model

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)

Figure 6.39.: Assigning the type of patient

supplied file for all the details.

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"/>

Figure 6.40.: Using the QUEUE module to rank order the queue

The balking and reneging logic have been placed inside two different sub-models. Balking only occurs for non-urgent patients, so the type of patient is first checked. If the patient is of type non-urgent, 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. This is illustrated in Figure 6.41.

The reneging logic is more complicated. Figure 6.42 presents an overview of the sub-model logic for reneging.

In the logic, the patient's type is checked to see if the patient is of type non-urgent. If so, the entity is sent to a SEPARATE module. The original entity simply exits the sub-model (to pro-

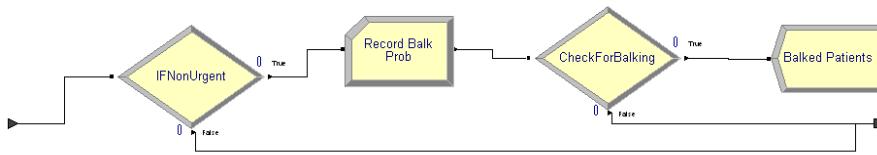


Figure 6.41.: Balking logic sub-model

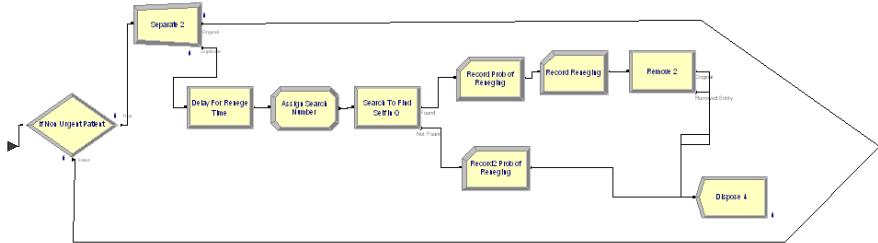


Figure 6.42.: Reneging logic sub-model

ceed 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 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.

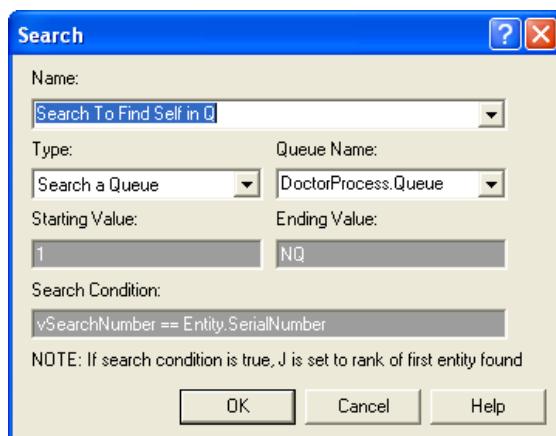


Figure 6.43.: SEARCH module for reneging logic

As illustrated in Figure 6.43, 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

6. Modeling Systems with Advanced Process Concepts

the search condition is true the global variable, J , is set to 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 6.44.

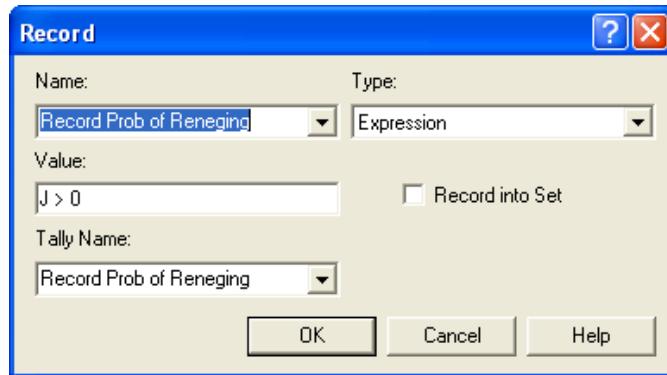


Figure 6.44.: Recording whether reneging occurred

If $J > 0$, then the entity at rank J can be removed from the `DoctorProcess.Queue` as illustrated in Figure 6.45. The rest of the model is relatively straightforward and you are encouraged to explore the final dialog boxes.



Figure 6.45.: REMOVE module for removing reneging patient

Assuming that the clinic opens at 8 am and closes at 6 pm, the simulation was set up for 30 replications to yield the results shown in Figure 6.46. It appears that there is a relatively high chance (about 29%) that a non-urgent 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.

Tally		
Expression	Average	Half Width
Record Balk Prob	0.00222222	0.00
Record Prob of Reneging	0.2884	0.07

Figure 6.46.: Example output for walk-in clinic model

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.

6.5. Holding and Signaling Entities

In this section we will 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: 1) wait for signal, 2) infinite hold, and 3) 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 (that 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

6. Modeling Systems with Advanced Process Concepts

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 2, 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 Figure 2.73 of Chapter 2, the entities are in the condition delayed state, waiting on the condition of particular signal. The default discipline of the queue is first in, first out. 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 2. 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 Section 6.6.3.

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.



In my opinion, modelers that use the scan for condition option are ill-informed. You can always replace a scan for condition HOLD module with the wait and signal option if you can identify where the condition changes. This is *always* more efficient. The only time that I would use a the scan for condition option is if there were many (more than 5) locations in the model where the condition could change. Do not fall in love with the scan for condition option. If you *know* what you are doing, you almost never need to use the scan for condition option and using the scan for condition option almost always indicates that you do not know what you are doing!

6.5.1. Redoing the M/M/1 Model with HOLD/SIGNAL

In this section, we will 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 queueing 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.

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 first in, first out rule.

6.5.1.1. First Solution to M/M/1 with HOLD/SIGNAL Example

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 the following pseudo-code.

```

CREATE customer every EXPO(1)
ASSIGN myArriveTime = TNOW
DECIDE
  IF vServer == 1
    BEGIN HOLD Server Queue
      Wait for End Service Signal
    END HOLD
  ENDIF
END DECIDE
ASSIGN vServer = 1
DELAY for EXPO(0.7)
ASSIGN vServer = 0
DECIDE
  IF Server Queue is not empty
    BEGIN SIGNAL End Service
      Signal Limit = 1
    END SIGNAL
  ENDIF
END DECIDE
RECORD TNOW - myArriveTime
DISPOSE

```

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

6. Modeling Systems with Advanced Process Concepts

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.

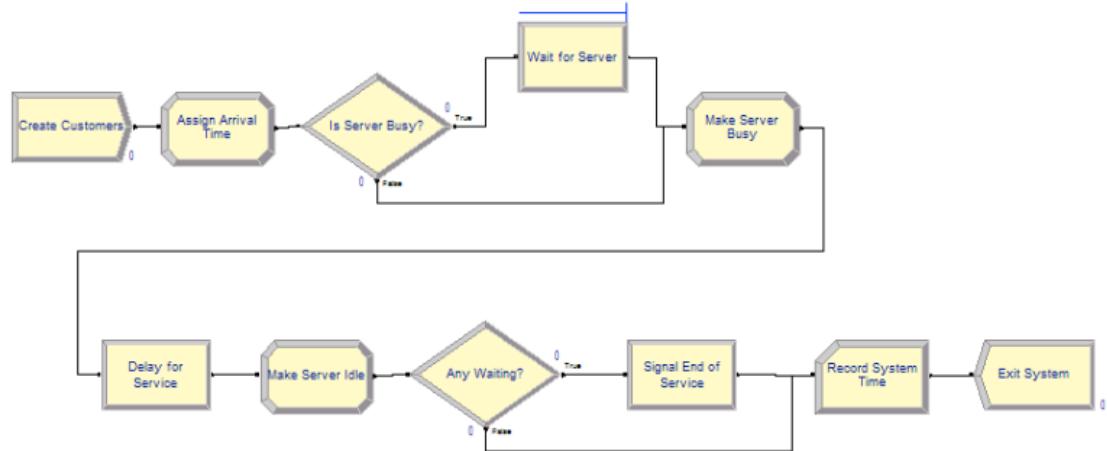


Figure 6.47.: First implementation of M/M/1 system with HOLD and SIGNAL

Figure 6.47 illustrates the flow chart for the implementation. The model represents the pseudo-code almost verbatim. The new HOLD and SIGNAL modules are shown in Figure 6.48 and Figure 6.49, respectively. The completed model can be found in file *FirstMM1HoldSignal.doe* found in the chapter files.

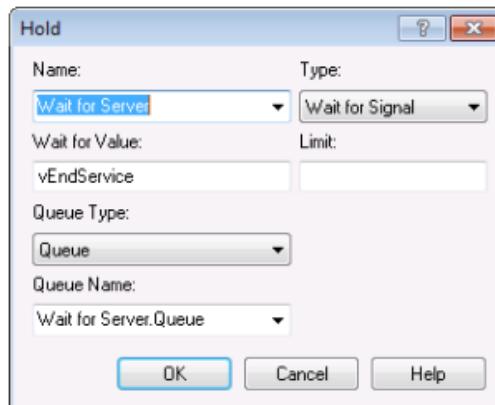


Figure 6.48.: HOLD module for first M/M/1 implementation

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 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.

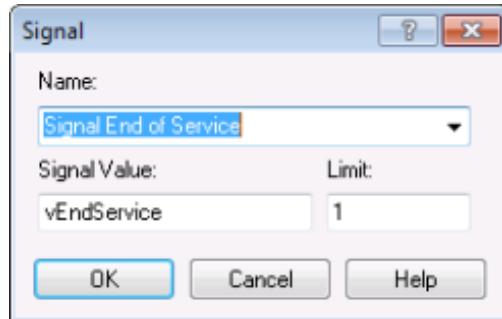


Figure 6.49.: SIGNAL module for first M/M/1 implementation

Figure 6.49 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 1 customer at a time, the limit value is set to 1. This causes 1 customer to be released from the HOLD, whenever the signal is sent.

Figure 6.50 presents the result from running the model for 30 replications with warm up 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.

Waiting Time		
	Average	Half Width
Wait for Server.Queue	2.3265	0.03
Other		
Number Waiting		
	Average	Half Width
Wait for Server.Queue	1.6261	0.03
User Specified		
Tally		
Interval		
	Average	Half Width
Record System Time	2.3264	0.02
Time Persistent		
Variable		
	Average	Half Width
vServerBusy	0.6994	0.00

Figure 6.50.: Results for first M/M/1 implementation

There is one key point to understand in 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

6. Modeling Systems with Advanced Process Concepts

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 (because they would all be waiting for the signal that is never coming). 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.

6.5.1.2. Second Solution to M/M/1 with HOLD/SIGNAL Example

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 6.51 presents the model for this second implementation. Note the two process flows in the figure. The file *SecondMM1HoldSignal.doe* contains the details.

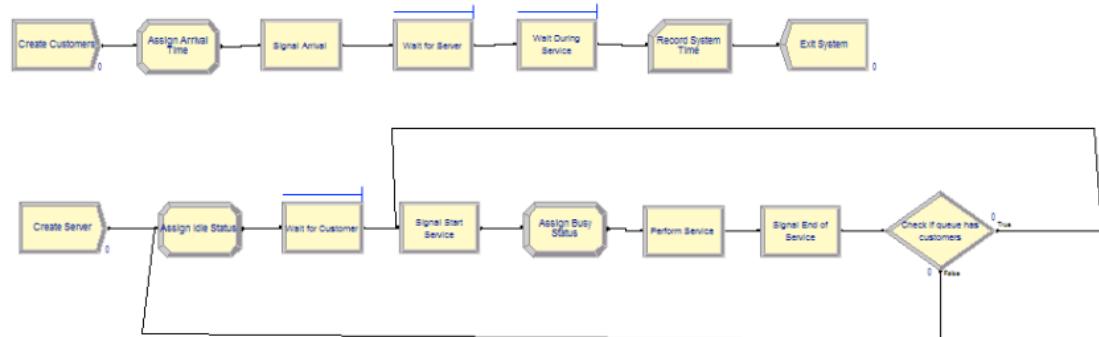


Figure 6.51.: Second implementation of the M/M/1 with HOLD and SIGNAL

The logic follow very closely the pseudo-code provided in the following pseudo-code.

```

CREATE customer every EXPO(1)
ASSIGN myArriveTime = TNOW
SIGNAL server of arrival
    Signal Value = vArrivalSignal

```

```

END SIGNAL
HOLD in Server Queue
    Wait for Begin Service Signal
END HOLD
HOLD in Service Queue
    Wait for End Service Signal
END HOLD
RECORD TNOW - myArriveTime
DISPOSE of customer

CREATE 1 server at time 0.0
LABEL: A
ASSIGN vServerStatus = 0
HOLD Wait for Customer arrival signal
    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
        GOTO to LABEL B:
    ELSE
        GOTO to LABEL A:
    ENDIF
END DECIDE

```

In the pseudo-code, 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 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. This second

6. Modeling Systems with Advanced Process Concepts

hold queue provides a place for the customer 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 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.

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.



Consider modeling resources as entities. This allows logic to be used to model how the resource behaves given the state of the system. This is an excellent way to model resources as intelligent agents.

6.5.2. Using Wait and Signal to Release Entities

In many situations, entities will need to wait until a specific time or condition occurs within a system. One classic example of this is order picking waves within a distribution facility. In this situation, orders (the entities) will arrive throughout the day and be held until a particular time that they can be released. The release of sets of the orders is timed to correspond to particular periods of time. In the parlance of distribution center picking operations, these periods are called waves, like waves hitting a beach.

The length and timing of the waves is typically determined by some optimization algorithm that ensures that the picking operations can be completed in an optimal manner in time for the subsequent shipping of the items. In addition, each order is assigned (again according to some optimization method) to a particular period (wave). For simplicity, we will assume that the periods (waves) are one hour long and that the orders are randomly assigned to each wave. Again, these assignments are often much more complex.

In this section, we will examine how to release picking orders within a distribution center under the following assumptions:

- Assume that at the beginning of a shift there are a number of orders that need to be picked within the warehouse
- Let N be the number of orders to pick
- Assume that $N \sim \text{Poisson}$ (mean rate = 100 orders/day)
- Assume that the warehouse has 4 picking periods (1, 2, 3, 4)
 - The first picking period starts at time 0.0
 - Each picking period lasts 60 minutes
- Assume that the orders are equally likely to be assigned to any of the four periods
- After the orders are released, they proceed to pickers who pick the orders and send them to be shipped.
 - Assume that the time to pick the entire order is $\text{TRIA}(5, 10, 15)$ minutes
- Build an Arena model to simulate this system over the picking periods
 - Determine the number of pickers needed to satisfactorily complete the picking operations.

The purpose of this example is to illustrate the basic ideas of using the wait and signal functionality of Arena to release the orders. The basic strategy will be to assign to each order a number that indicates when it should be released. In other words, we assign its release period (or wave). Then, the arriving order will wait in a HOLD module for the appropriate signal representing the start of its period. At which time, the order will be released to pickers for processing.

The first modeling issue is to represent the creation of the orders. In a typical distribution center setting the customer orders may arrive throughout the day; however, they are typically not processed immediately upon arrival. In most situations, they are processed by the order processing systems of the distribution center (human or computer) and made ready for release. There is typically a random number of orders to be processed. For simplicity in this example, we will assume that the number of orders per day is well modeled with a Poisson distribution with a mean rate of 100 per day. Thus, at the beginning of the day all orders are available and assigned their release period. Let's assume that there will be four release periods, each spaced 1 hour apart with the first period starting at time 0.0 and lasting one hour. Thus, the second period starts at time 1.0, and lasts an hour, etc. We will number the (waves) periods 1, 2, 3, 4.

Because of our assumption that orders are randomly assigned to waves, we need a distribution to represent this assignment. To keep it simple, we will assume that the orders are equally likely to be assigned any of the periods 1, 2, 3, 4. Thus, we need to generate a discrete uniform random variate over the range from 1 to 4. Recall that the Arena formula for generating a $\text{DU}(a,b)$ random variate, X, is:

$$X = a + \text{AINT}((b - a + 1) * \text{UNIF}(0,1))$$

6. Modeling Systems with Advanced Process Concepts

Thus, to generate DU(1,4), we have,

```
1 + AINT((4 - 1 + 1)*UNIF(0,1)) = 1 + AINT(4*UNIF(0,1))
```

Again, in general, the assignment of orders to particular picking periods (waves) is often a much more complex process that takes into account the order due dates, the work associated with the orders, and the locations of the items to be picked.

Let's take a look at the pseudo-code to create and hold the orders for later release.

```
ATTRIBUTES: myWave // to hold which picking wave to listen for
--
CREATE POIS(100) entities with 1 event at time 0.0
ASSIGN myWave = 1 + AINT(4*UNIF(0,1))
HOLD wait for signal myWave
Go To: Label Picking Process
```

That's it! This will create the orders, assign their wave number and cause them to wait within a HOLD module for a signal that is specified by the value of the attribute, myWave. When this signal (number) is broadcast by a SIGNAL module all the entities (orders) waiting in the HOLD queue for that corresponding wave number will be released and proceed for further processing.

The second modeling issue is to represent the timing and signaling of the picking waves. To implement this situation, we will create a “clock-entity” that determines the period and makes the signal. For this situation we will use a global variable, vPeriod, that keeps track of the current period and is used to signal the start of the period.

Now, we have one tricky issue to address. The start of the first wave happens at time 0.0, but as noted in the previous pseudo-code, the orders for the day are also created at time 0.0. We need to coordinate between two CREATE modules to ensure that the clock-entity allows the orders to be created first. We must ensure that the orders are in the HOLD module before the signal for the first wave occurs; otherwise, the orders associated with the first wave will not be in the HOLD queue when the signal occurs. Thus, they will miss the first signal and not be released.

While there are number of ways to address this issue, the simplest is to cause the clock-entity to be created infinitesimally after time 0.0. This will allow all the entities (orders) from the order creation process to be created and go into the HOLD module queue before the clock-entity signals the start of the first wave. Let's take a look at the pseudo-code for this situation.

We will use three variables to represent the time periods.

```
VARIABLES:
vPeriod = 0 // to track the current period
vPeriodLength = 60 // minutes per period
vLastPeriod = 4 // the number of periods
```

Then, we will use a logical entity, call it the *clock-entity*, that will loop around signaling the appropriate period. In essence, it is ticking off the periods through time. This is a very common modeling pattern.

```

CREATE 1 entity at time 0.0001
A:
ASSIGN vPeriod = vPeriod + 1
If vPeriod \> vLastPeriod
DISPOSE
ELSE
SIGNAL vPeriod
DELAY vPeriodLength minutes
Go to: Label A

```

Notice that in this pseudo-code there are two defined variables (`vPeriodLength` and `vLastPeriod`) to make the code more generic. Using these variables, we can easily vary the number of periods and the length of each period. Notice also that the CREATE module indicates that the clock entity is created at time 0.0001. This ensures that the CREATE module associated with creating the orders goes first (at time 0.0) and allows the orders to fill up the HOLD queue. The clock-entity then increments the variable, `vPeriod`, so that the current period (wave) is noted. We then check if the current period is greater than the last period to simulate. If so, the clock-entity is disposed.

If the last period has not happened, we need to signal the start of the period. When the clock-entity executes the SIGNAL module, the value of the signal (`vPeriod`) is broadcast globally throughout the model. *Any* entities in *any* HOLD modules that are waiting for that particular number as their signal will be removed from the HOLD queue and placed in the ready state to be processed at the current time. Since the clock-entity immediately enters a DELAY module, it is placed in the time-delayed state to become the active entity 60 minutes later. Thus, all the entities that were signaled can now be processed. According to our previous pseudo-code, they will go to the label Picking Process.

As noted previously, there are a number of other ways to coordinate the timing of the order process and the wave process to ensure that the orders are placed in the HOLD queue before the clock-entity makes the first signal. Rather than using an infinitesimal delay as illustrated here, you could remove the CREATE module for the clock-entity. To ensure that the clock-entity is made after the orders are made, you can have the order making process create the clock-entity. One way to do this is to have the first order created go through a SEPARATE module. The duplicate entity will be the clock entity and it would be sent to the logic that signals the waves (pseudo code labeled A). Because a SEPARATE module places the duplicate entities in the ready state, the clock-entity will not proceed until the orders are all waiting. The interested reader is encouraged to test their knowledge by implementing this approach. The implementation and analysis of this example within Arena is left as an exercise.

6. Modeling Systems with Advanced Process Concepts

In the next section, we see how essential the wait and signal constructs are to the modeling of an important situation within inventory systems.

6.5.3. Modeling a Reorder Point, Reorder Quantity Inventory Policy

In an inventory system, there are units of an item (e.g. computer printers, etc.) 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. back ordered). 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: 1) When to order? and 2) How much to order?. If the system does not order enough or does not order at the right time, the system not be able to fill customer demand in a timely manner. Figure 6.52 illustrates a simple inventory system.

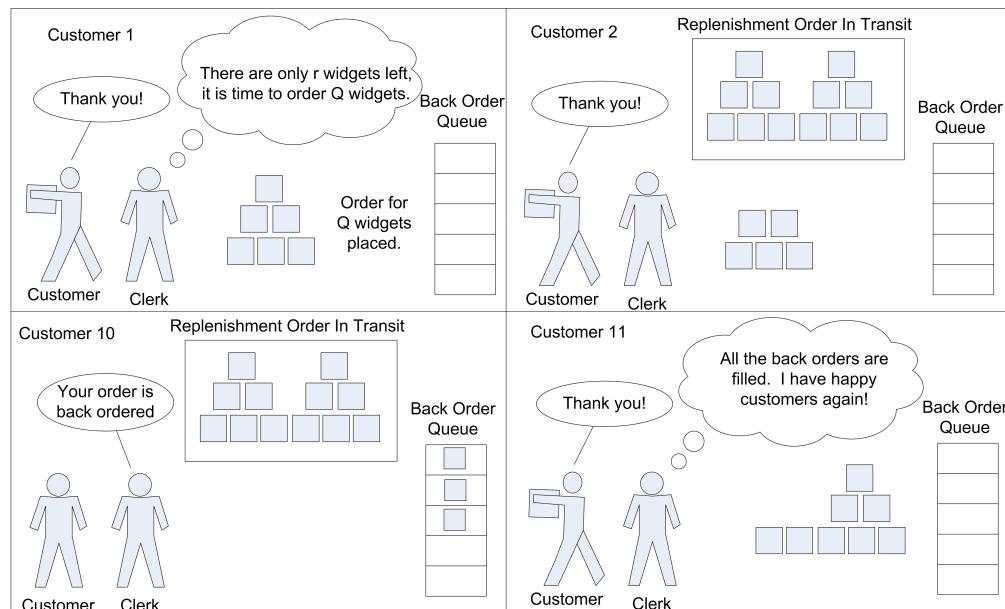


Figure 6.52.: A simple reorder point, reorder quantity inventory system

There are a number of different ways to manage the replenishment process for an inventory item. These different methods are called *inventory control 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.

The topic of inventory systems has been well studied. A full exposition of the topic of inventory systems is beyond the scope of this text, but the reader can consult a number of texts within the area, such as (Hadley and Whitin, 1963), (Axssäter, 2006), Silver et al. (1998), or (Zipkin, 2000) for more details. The reader interested in supply chain modeling might consult (Nahmias, 2001), (Askin and Goldberg, 2002), (Chopra and Meindl, 2007), or (Ballou, 2004).

Within this section we will develop a model of a continuous review (r, Q) inventory system with back-ordering. 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 back-ordered. The back-orders are accumulated in a queue and they will be filled after the arrival of a replenishment order. Assume for simplicity that the back-orders are filled on a first come first served basis. The inventory position (inventory on-hand plus on-order minus back-orders) is checked each time after a regular customer demand and the occurrence of a back-order. 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 back-orders and increase the on-hand inventory. The time from when a replenishment order is placed until the time that it arrives to fill any back-orders 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 back-ordered, 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 both 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 re-order point as demands arrive to be filled. After filling (or back-ordering) 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 back orders. The pseudo-code for this situation is given the following pseudo-code.

6. Modeling Systems with Advanced Process Concepts

```

CREATE demand
ASSIGN myAmtDemanded = 1
Send to LABEL: Order Fulfillment

LABEL: Order Fulfillment
DECIDE
    IF I(t) >= myAmtDemanded THEN //fill the demand
        ASSIGN I(t) = I(t) - myAmtDemanded
    ELSE // handle the backorder
        ASSIGN BO(t) = BO(t) + myAmtDemanded
        SEPARATE Duplicate 1 entity
            Send duplicate to LABEL: Handle Back Order
    ENDIF
    ASSIGN 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) - myAmtDemanded
Send to LABEL: Order Fulfillment

LABEL: Replenishment Ordering
IF IP(t) <= r
    ASSIGN IO(t) = IO(t) + Q //place the order
    DELAY for lead time
    ASSIGN IO(t) = IO(t) - Q //receive the order
    ASSIGN I(t) = I(t) + Q
    SIGNAL: Hold back order queue
ENDIF
DISPOSE

```

Referring to the pseudo-code, 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 $I(t) \geq myAmtDemanded$. Then, the on hand inventory is decremented by the amount of the demand. If the demand cannot be satisfied ($I(t) < myAmtDemanded$). 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 back order 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 back-order queue. A signal is sent to the back order queue so that if there are any demands waiting in the back order 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 back-ordered, the percentage of time that the system is out of stock, and the average number of orders made per unit time.

Let's discuss these performance measures before indicating how to collect them within a simulation. The average inventory on hand and the average amount of inventory back-ordered 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-persistent 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:

$$SO(t) = \begin{cases} 1 & I(t) = 0 \\ 0 & I(t) > 0 \end{cases}$$

$$\bar{SO} = \frac{1}{T} \int_0^T SO(t) dt$$

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 \bar{SO} can be interpreted as the proportion 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).

6. Modeling Systems with Advanced Process Concepts

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 back ordered 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 back order cost for the item in terms of \$/unit/time. That is, for every unit of inventory back ordered, 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:

$$\overline{TC} = k\overline{OF} + h\bar{I} + b\overline{BO}$$

A discussion of the technical issues and analytical results related to these variables can be found in Chapter 6 of (Zipkin, 2000). Let's take a look at an example that illustrates an inventory situation.

Example 6.3. 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 back-order 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:

1. Average inventory on hand and back ordered
 2. Average frequency at which orders are placed
 3. Probability that an arriving customer does not have their demand immediately filled
 4. Average total cost per time
-

The Arena model will follow closely the previously discussed pseudo-code. To develop this model, first define the variables to be used as per Figure 6.53. 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 back-ordered inventory. This will cause time-based statistics to be collected on these variables.

Variable - Basic Process							
	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 6.53.: Defining the inventory model variables

Figure 6.54 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 $(1.0/0.12)$ days (3.6 units/month * (1 month/30 days) = 0.12 units/day).

The ASSIGN module in Figure 6.55 shows the amount demanded set to 1 and a stock out indicator set to zero. This will be used to tally the probability that an incoming demand is not filled

6. Modeling Systems with Advanced Process Concepts

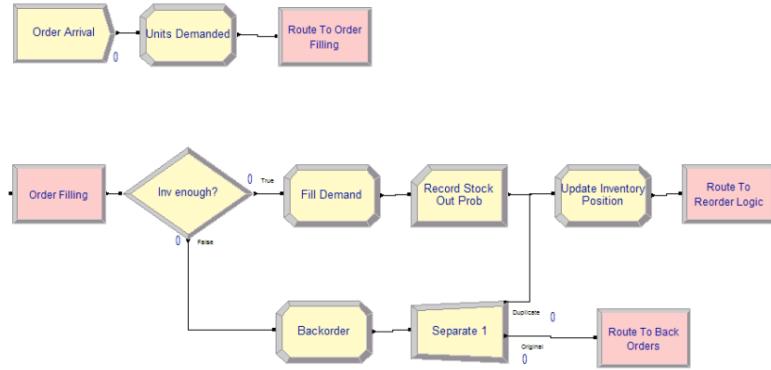


Figure 6.54.: Initial order filling logic for the (r, Q) inventory model

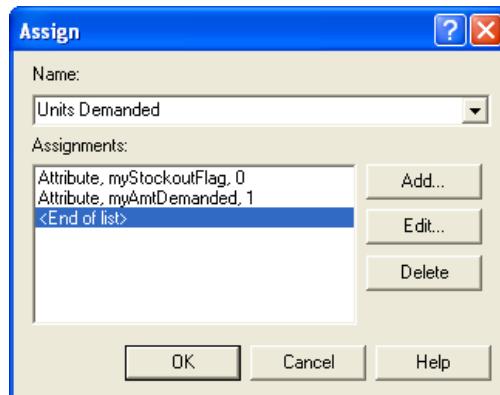


Figure 6.55.: Assigning the amount of demand

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 back-ordered. The ASSIGN module labeled Back-order increments the number of back orders. Then the SEPARATE module creates a duplicate, which goes to the station than handles back orders. 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 6.56. The amount on hand or the amount back-ordered is decreased or increased accordingly. In addition, the inventory position is updated. In the back-ordering AS-

6.5. Holding and Signaling Entities

SIGN module, the stock out flag is set to 1 to indicate that this particular demand did not get an immediate fill.

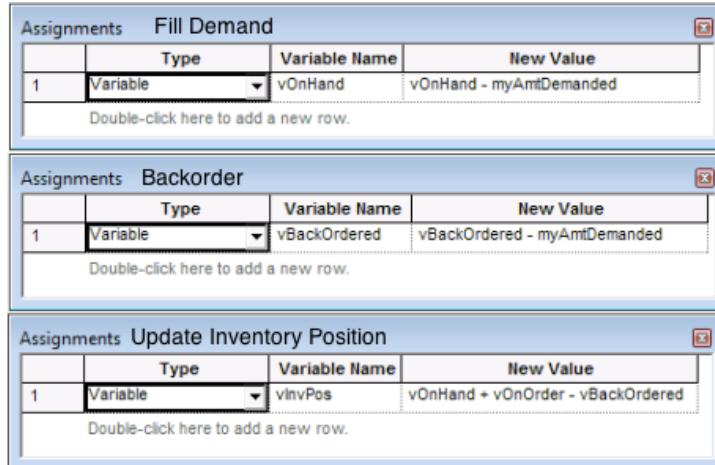


Figure 6.56.: ASSIGN modules for filling, backordering, and updating inventory position

When the demand is ultimately filled, it will pass through the RECORD module within Figure 6.54. 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 back-ordered upon arrival and the value 0 if it had not been back-ordered upon initial arrival. This indicator variable will estimate the probability that an arriving customer will be back-ordered. 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 back-ordered will be the same as \overline{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))

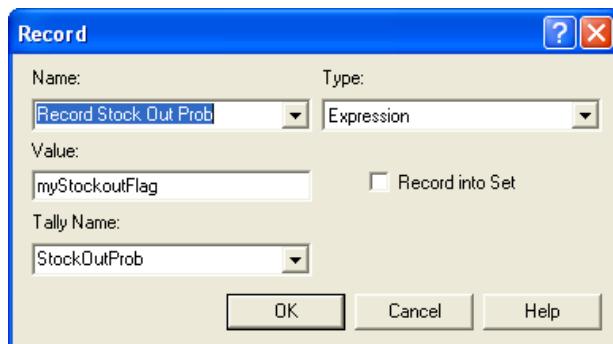


Figure 6.57.: Recording the initial fill indicator variable

The logic representing the back ordering and replenishment ordering process is given in Figure 6.58. Notice how STATION modules have been used to denote the start of the logic. In the

6. Modeling Systems with Advanced Process Concepts

case of back ordering, the Handle Back Order station has a HOLD module, which will hold the demands waiting to be filled in a queue. See Figure 6.59. When the HOLD module is signaled (value = 1), the waiting demands are released. The release will be triggered after a replenishment order arrives. The number of waiting back orders is decremented in the ASSIGN module and the demands are sent via a ROUTE module for order fulfillment.

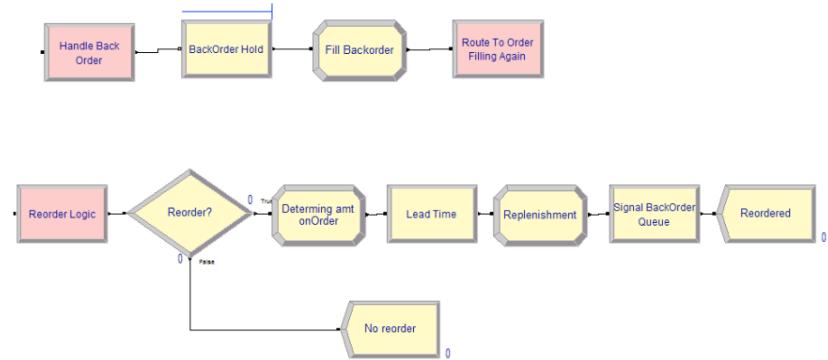


Figure 6.58.: Back ordering and replenishment logic

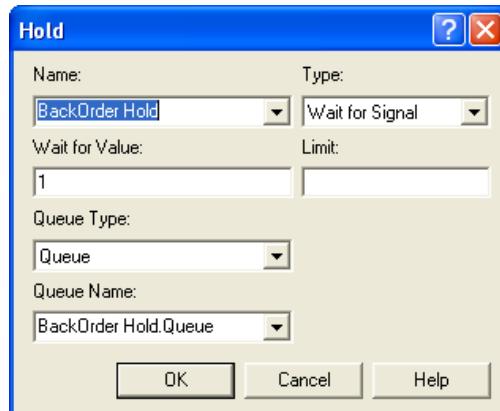


Figure 6.59.: Backorder queue as a HOLD module

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 Figure 6.60 and Figure 6.61, 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 $\bar{OF} = N(T)/T$. 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 will ob-

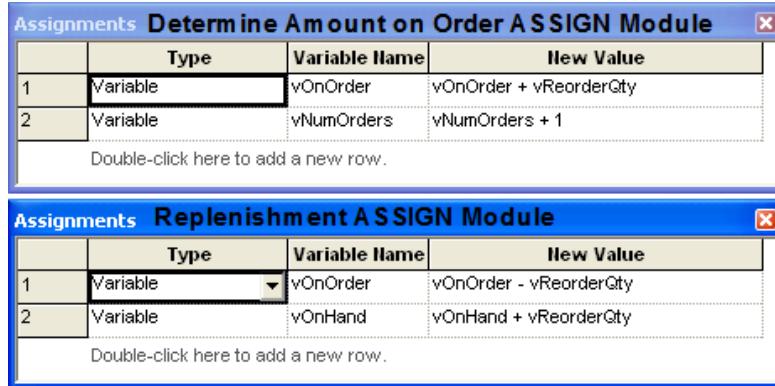


Figure 6.60.: Ordering and replenishment ASSIGN modules

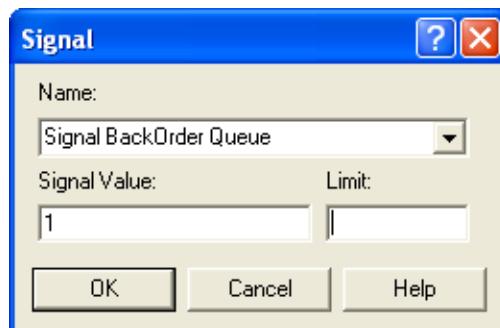


Figure 6.61.: Signalling the backorder queue

serve the number of orders that have been placed since the start of the observation period. In Figure 6.54, there is a variable called, `vNumOrders`, which is incremented each time an order is placed. This is, in essence, $N(T)$. Figure 6.62 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).

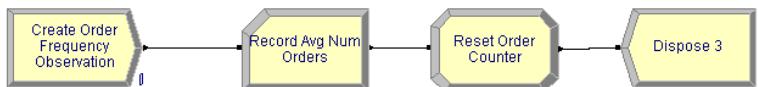


Figure 6.62.: Order frequency collection logic

Then the RECORD module, shown in Figure 6.63, 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 \bar{SO} as well as the cost of the policy needs to be discussed. To collect \bar{SO} , a time-persistent statistic needs to be defined using the Statistic module of the Advanced Process panel as in Figure 6.64. The expression `(vOnHand == 0)` is a boolean expression, which evaluates to 1.0 if true and 0.0 if false.

6. Modeling Systems with Advanced Process Concepts

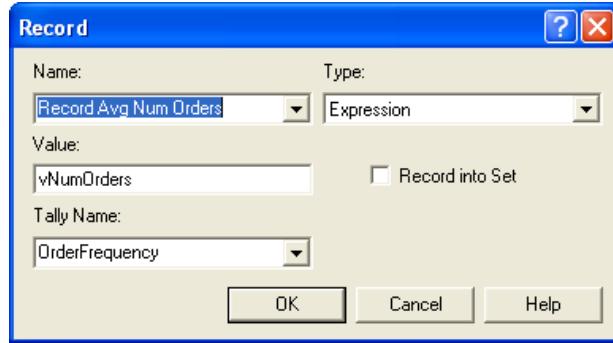


Figure 6.63.: Recording the number of orders placed

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.

Statistic - Advanced Process					
	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 6.64.: 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 back-ordering 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 back-ordering cost, the holding cost, and the ordering cost. In Figure 6.64, 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 reports as user-defined statistics. The completed model is found in file `rQInventoryModel.doe` in the chapter files.

The case of $(r = 1, Q = 2)$ was run for 30 replications with a warm up period of 3600 days and a run length of 39,600 days. Figure 6.65 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 Poisson arrivals is working according to theory.

This example should give you a solid basis for developing more sophisticated inventory models. While analytical results are available for this example, small changes in the assumptions neces-

Expression	Average	Half Width
OrderFrequency	1.8022	0.01
StockOutProb	0.4034	0.01
Time Persistent		
Time Persistent	Average	Half Width
TimeOutOfStock	0.4034	0.00
Variable		
Variable	Average	Half Width
vBackOrdered	0.2952	0.01
vOnHand	0.9926	0.01
Output		
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 6.65.: Results for the (r, Q) inventory model

sitate 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 back-order queue should be done in different ways. For example, suppose customers wanting 5 and 3 items respectively were waiting in the queue. Now suppose a replenishment of 4 items comes in. Do you give 4 units of the item to the customer wanting 5 units (partial fill) or do you choose the customer wanting 3 units and fill their entire back-order 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, we will expand our modeling capabilities by looking at a number of useful constructs related to how entities behave within a model.

6.6. Miscellaneous Modeling Concepts

This section discusses a number of additional constructs available within 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 than the functionality available in the BATCH module.

6.6.1. Picking Between Stations

Imagine that you have just finished your grocery shopping and that you are heading towards the check-out area. You look ahead and scan the checkout 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, least busy resource, etc.). Because of this 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 waiting at the stations. After receiving processing at the stations, the entities then depart the system.

As can be seen in Figure 6.66, 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.

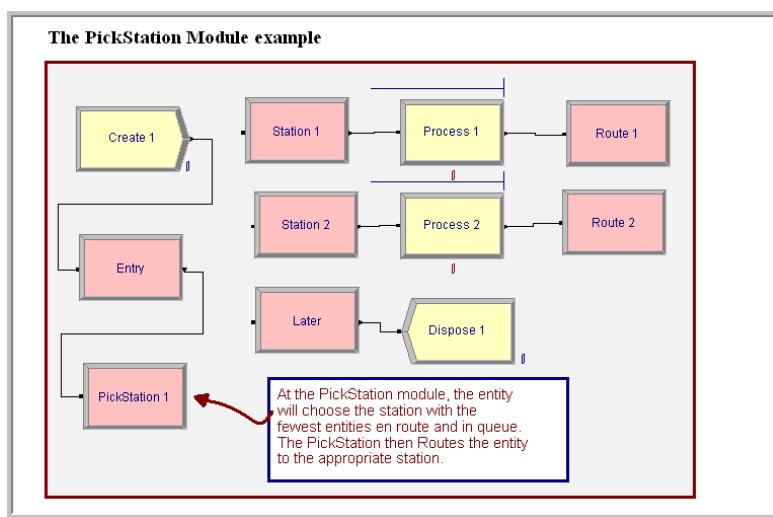


Figure 6.66.: Overview of PICKSTATION example

The PICKSTATION module is very similar to a ROUTE module as shown in Figure 6.67. 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

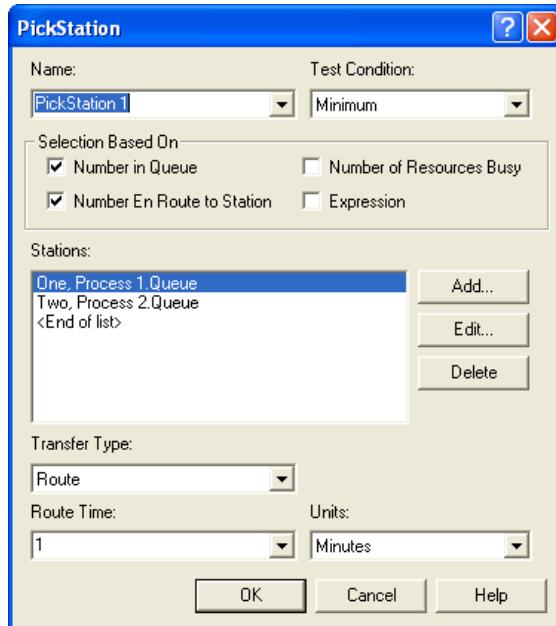


Figure 6.67.: PICKSTATION module

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 6.68.: Adding a station to the PICKSTATION module

Figure 6.68 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

6. Modeling Systems with Advanced Process Concepts

evaluated based on the station. scans the list to find the station that has the minimum (or maximum) according to the criteria selected. In the case of ties, will pick the station that is listed first according the Stations list. After the station has been selected, the entity will be routed according to the standard route options (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 straight forward. The only difficulty lies in properly using the Expression option.

Let's consider a modification to the example to illustrate the use of an expression. Let's 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 6.69 shows the two schedules. In the schedule for resource 1, the capacity is 0 for the first 10 minutes and is then 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.

	Value	Duration
1	0	10
2	1	50

	Value	Duration
1	1	30
2	0	10
3	1	20

(a) (a) Schedule for resource 1 (b) (b) Schedule for resource 2

Figure 6.69.: Schedules for PICKSTATION extension for resource 1 and 2

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: `(STATE(resource name) == INACTIVE_RES)*vBigNumber`. In this case, if the named resource's

state is inactive the Boolean expression yields a 1 for true. Then the 1 is multiplied by a big number (e.g. 10000), which would yield the value 10000 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 6.70. 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.

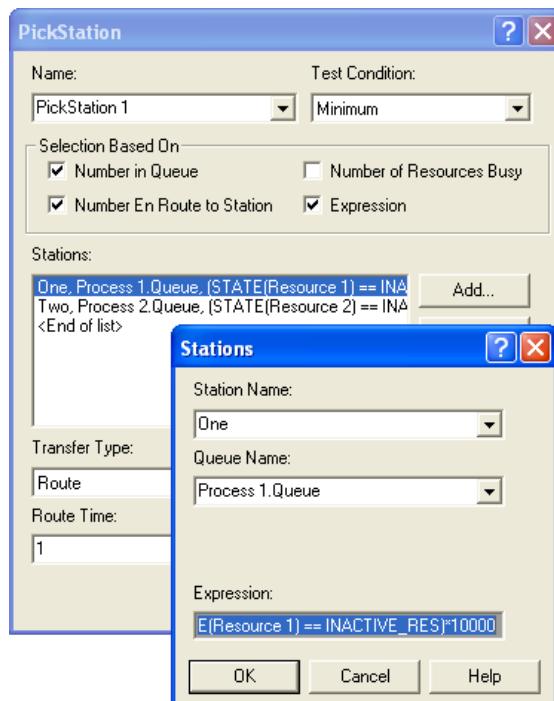


Figure 6.70.: Preventing a station from being chosen while inactive

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.

6.6.2. 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. 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 handles variables and attributes. The trick to converting a series of modules to generic stations is to properly index into sets. An example is probably the best way to demonstrate this idea.

6. Modeling Systems with Advanced Process Concepts

Let's reconsider the test and repair shop from Chapter 4 and shown in Figure 6.71. 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 9 modules with 4 modules that can handle any part intended for any of the test stations.

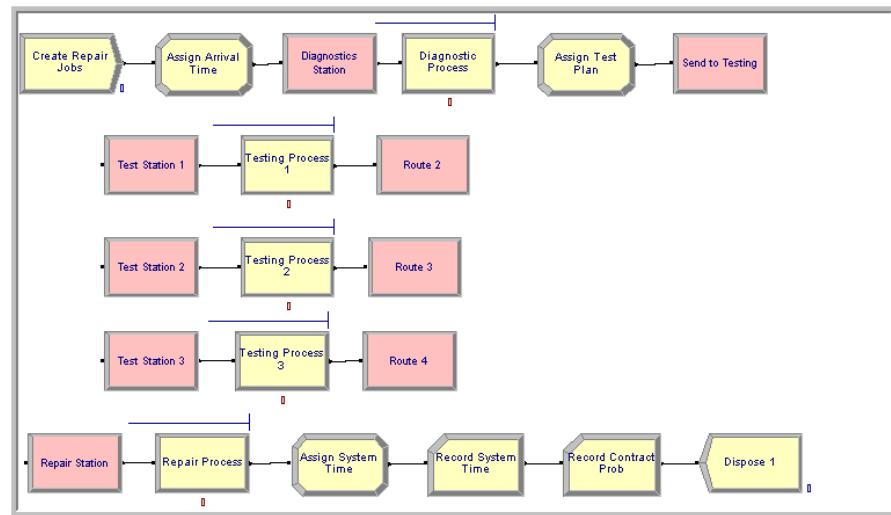


Figure 6.71.: Overview of test and repair model

Figure 6.72 show 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.

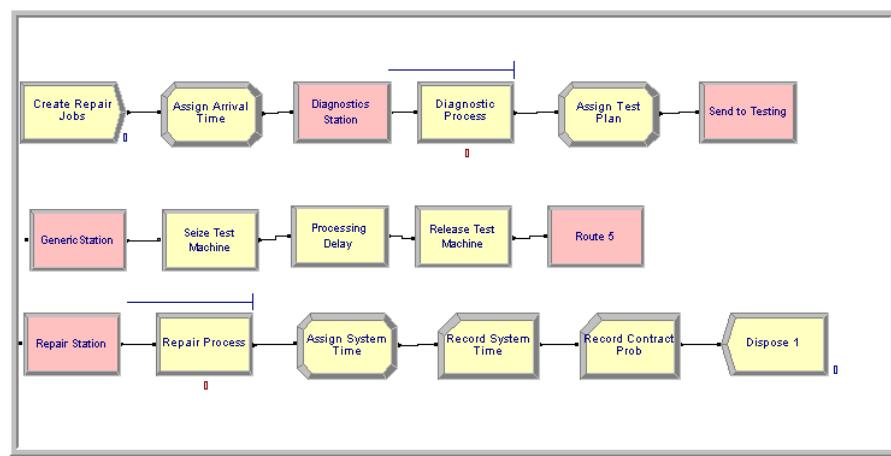


Figure 6.72.: Generic station version of test and repair model

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.

There are two ways to define a set of stations: 1) using the Advanced Set module or 2) 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 6.73 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. 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 that will be used, you need to make sure that the resources and queues associated with the stations are listed in the same order.

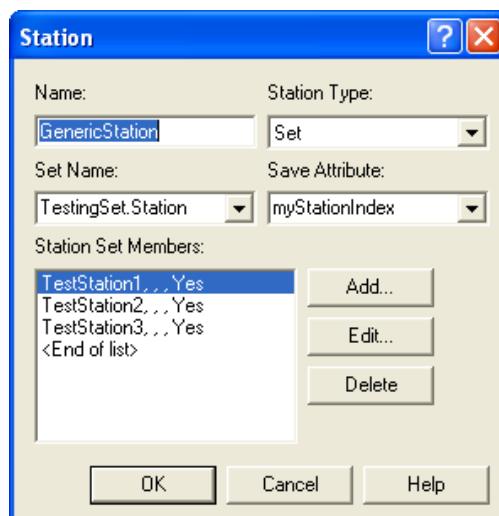


Figure 6.73.: Generic station module with set option

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's 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 6.74 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.

6. Modeling Systems with Advanced Process Concepts

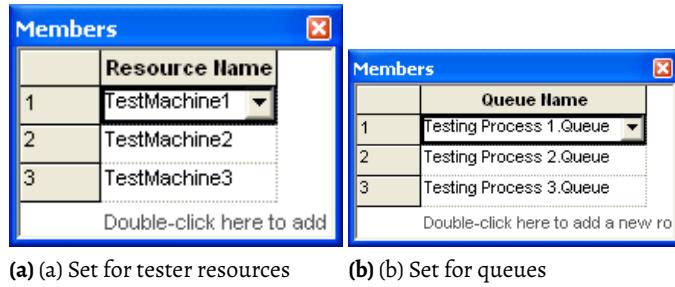


Figure 6.74.: Resource and queue sets for generic station modeling

Figure 6.75 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 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.

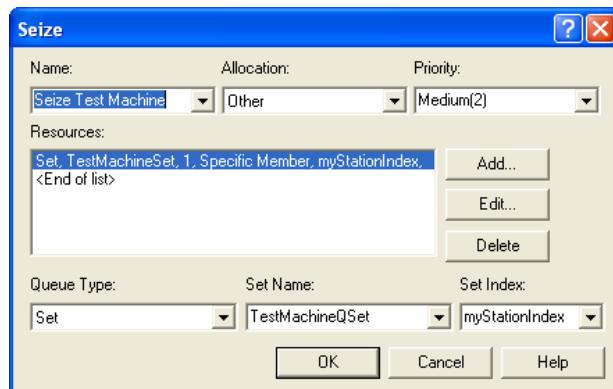


Figure 6.75.: SEIZE module for generic station model

The DELAY module for the generic station is shown in Figure 6.76. 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 6.77. 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.

If you run this model, found in the file `GenericRepairShop.doe`, the results will be exactly as seen

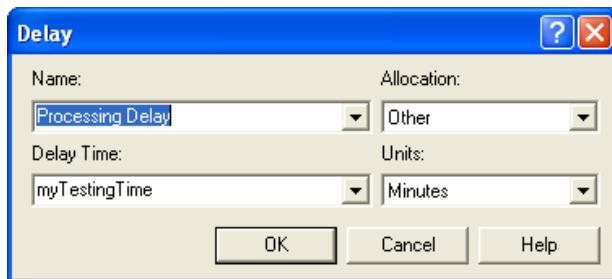


Figure 6.76.: DELAY module for generic station model

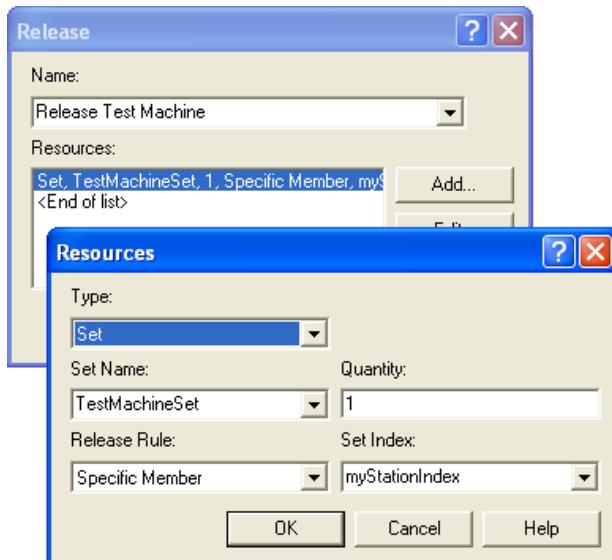


Figure 6.77.: RELEASE module for generic station model

in Chapter 4. Now, you might be asking yourself: "This generic stuff seems like a lot of work. Is it worth it?" Well, in this case 4 flow chart modules were saved (5 in the generic versus 9 in the original model). Plus, three additional sets were added in the generic model. This doesn't seem like much of a savings. Now, imagine a realistically sized model with many more test stations, e.g. 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 flow chart modules. All the changes that would need to take place to run a bigger model would be in data modules (e.g. sets, sequences, etc). 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

6. Modeling Systems with Advanced Process Concepts

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 don't 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.

6.6.3. Picking up and Dropping Off Entities

In many situations, entities must be synchronized so that they flow together within the system. Chapter 4 presented how the BATCH and SEPARATE 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 enter 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. Figure 6.78, 6.79, 6.80 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

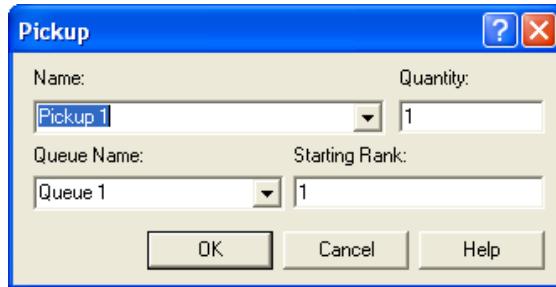


Figure 6.78.: The PICKUP module



Figure 6.79.: The DROPOFF module

up 1 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 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

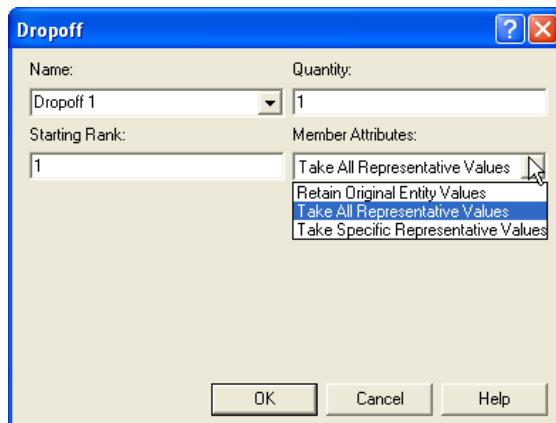


Figure 6.80.: The DROPOFF module dialog

6. Modeling Systems with Advanced Process Concepts

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 a 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 that 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's start with the modeling of the passengers.

Figure 6.81 presents an overview of the bus system model. The entire model can be found in the file *BusSystem.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 6.81 represent the creation of the bus. The bus entity is created at time zero 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.

6. Modeling Systems with Advanced Process Concepts

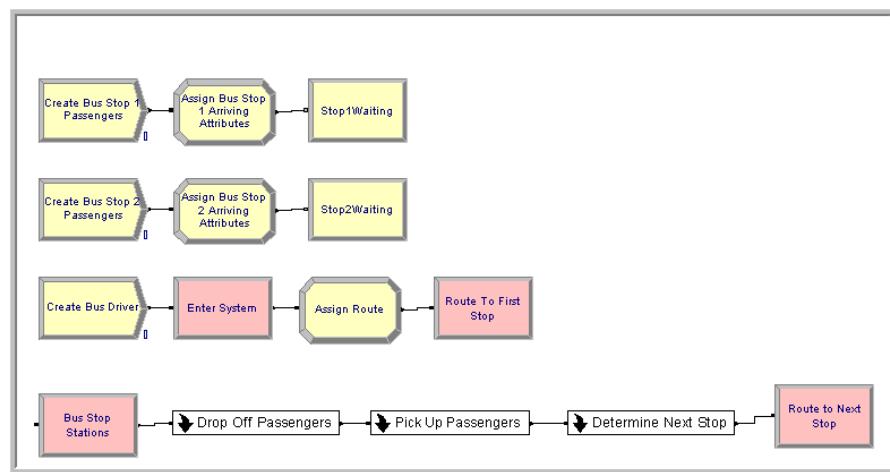


Figure 6.81.: Overview of the bus system model

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's discuss the details of the bus station modeling.

As previously mentioned, a generic station can be used to represent each bus stop. Figure 6.82 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 6.81, three sub-models have been used to represent the drop off logic, the pick up logic, and the logic to decide on the next bus stop to visit.



Figure 6.82.: Generic station for bus stops

The logic for dropping off passengers is shown in Figure 6.83. 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 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 6.84, is executed. Notice that in this case, since the system time of the passengers must be captured, the *Retain Original Entity Values* option should be used.

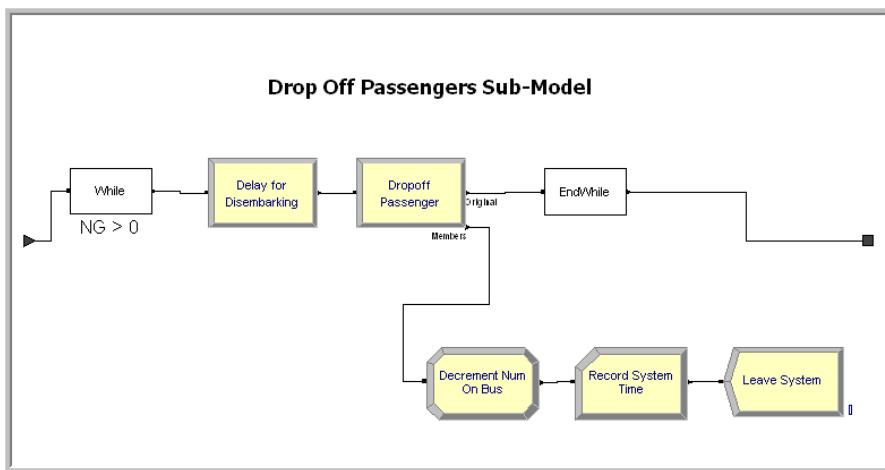


Figure 6.83.: Dropping off passengers sub-model

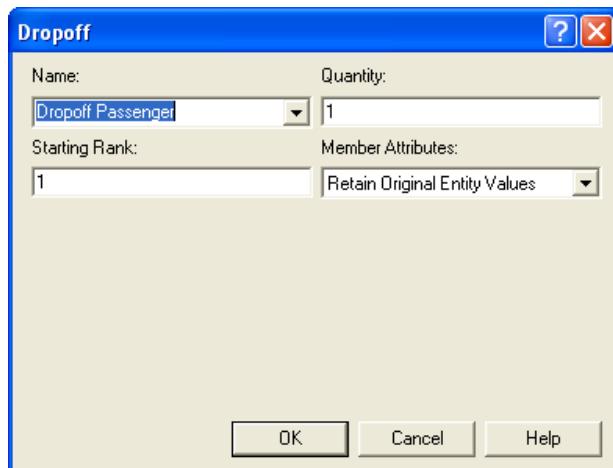


Figure 6.84.: DROPOFF module for passengers

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.

6. Modeling Systems with Advanced Process Concepts

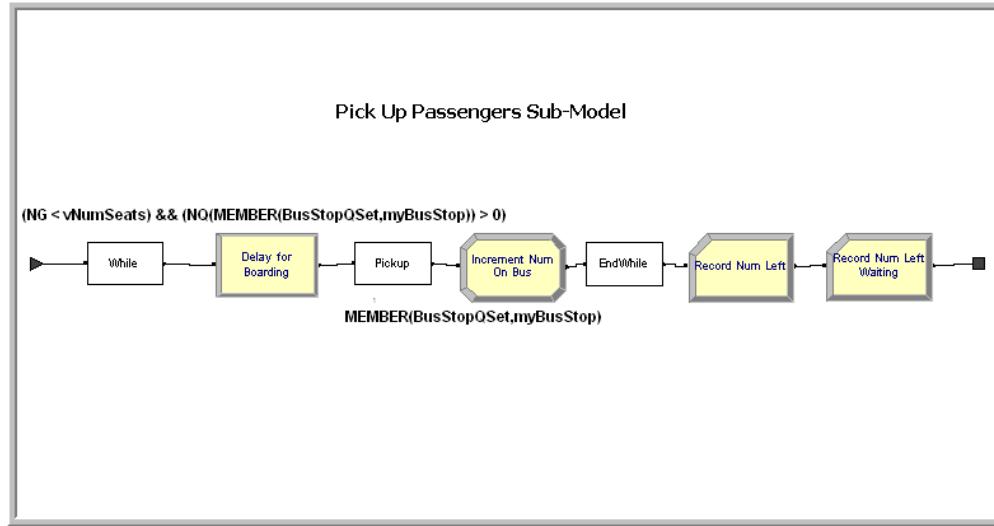


Figure 6.85.: Picking up passengers sub-model

The logic to pick up passengers is very similar to the logic for dropping off the passengers. In this case, a WHILE-ENDWHILE construct will again be used, as shown in Figure 6.85; however, there are a couple of issues that need to be handled carefully in this sub-model.

In the drop-off passenger sub-model, 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's 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 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 6.86, 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.

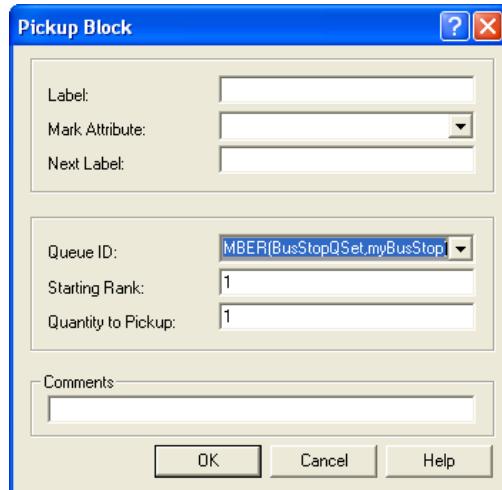


Figure 6.86.: The PICKUP Block

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 6.87 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))`.

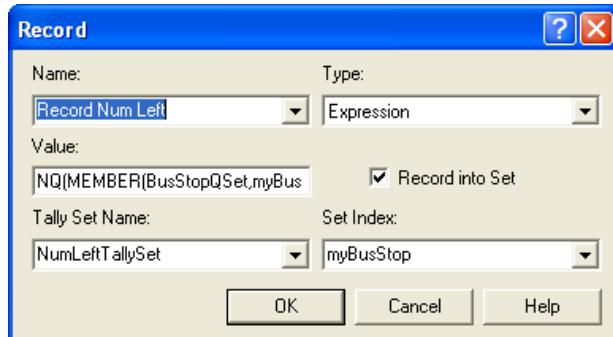


Figure 6.87.: Recording the number of passengers not picked up

Running the model for 10 replications of 8 hours yields the results shown in Figure 6.88. Notice that the waiting time for the passengers is a little over 20 minutes for the two queues. In addi-

6. Modeling Systems with Advanced Process Concepts

tion, the results indicate that on 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.

Waiting Time	Average	Half Width
Stop1Waiting.Queue	20.7487	1.24
Stop2Waiting.Queue	23.2777	2.90
Other		
Number Waiting	Average	Half Width
Stop1Waiting.Queue	2.0739	0.21
Stop2Waiting.Queue	3.9714	0.80
User Specified		
Tally		
Expression	Average	Half Width
Record Num Left Waiting	0.2567	0.29
Interval	Average	Half Width
Record System Time	42.9741	1.65
Usage		
None	Average	Half Width
NumLeftQ1	0.00	0.00
NumLeftQ2	0.5182	0.58

Figure 6.88.: Results for the bus system model

Based on this example, modeling with the PICKUP and DROPOFF modules is not too difficult. The additional functions, e.g. AG0 etc., 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 summarizes this chapter.

6.7. Summary

This chapter provided a discussion of miscellaneous modeling constructs that can improve your modeling. The modeling of non-stationary arrivals was discussed and it motivated the explo-

6.7. Summary

ration of advanced resource modeling constructs. The advanced resource modeling constructs allow for resources to be subjected to either scheduled or random capacity changes. Then, we learned about how the HOLD and SIGNAL constructs can allow for significant flexibility in representing complex control logic. Multiple examples of the usage of HOLD and SIGNAL modules were presented that can serve as the basis for very advanced modeling. Finally, a few interesting and useful miscellaneous modeling constructs were presented. For example, 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. With all of these concepts within your simulation toolkit, you are now prepared to model very advanced situations.

6.8. Exercises

Exercise 6.1. 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, 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 has 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 clinic according to non-stationary Poisson process. 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 log-normally 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 (use stream 6).

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 of our recommendation to within 2 minutes.

Exercise 6.2. 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 $j = 0, 1, 2$, etc, 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.

Exercise 6.3. 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 inter-arrival 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.

Suppose that there is limited space at the second station. In particular, there is room for 1 customer to be in service at the second station and room for only 1 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 it to wait at the second station. In other words, the customer will not release the server at the first station unless it can move to the second station.

Develop a model for this system using the STATION and ROUTE modules. Run the simulation for exactly 20000 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.

- a. Use a resource to model the waiting space at the second station. Ensure that a customer leaving the first station does not release its resource until it is able to seize the space at the second station.

6. Modeling Systems with Advanced Process Concepts

- 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. 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 for part (a) and rerun it under the same conditions as in part (a).
-

Exercise 6.4. 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 waiting in the queue until the server is available. The queue is processed according to a first in, first out rule. Use HOLD/SIGNAL to model this situation. Unlike in the second implementation of the M/M/1 illustrated in Section 6.5.1.2 develop your model so that the customer has the delay for service in its process flow.

Exercise 6.5. 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 a $1/14$ 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 back-ordering is very difficult to estimate, but a guess has been made that the annualized cost of a back-order is about \$25 per unit per year.

- a. Using 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 back-orders, and the probability of a stock out for your model.
 - b. 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 part (a).
 - c. Verify your results for part (a) using results from analytical inventory theory.
-

Exercise 6.6. 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 issues 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 6.7 presents 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 am to 8 pm 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 6.7, 6.8, (6.9), and (6.9). You can find this data in the chapter files that accompany this chapter in the spreadsheet called *Super Ready Auto Club.xlsx*.

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 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. The data for this exercise is available within the files associated with this chapter.

- a. What happens to the waiting times if road-side assistance calls are given priority over the credit card and travel service calls?
- 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).

Table 6.7.: 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

6. Modeling Systems with Advanced Process Concepts

Hour	Mon	Tue	Wed	Thur	Fri	Sat	Sun
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 6.8.: 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

Table 6.9.: 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 6.10.: 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

Exercise 6.7. The test and repair system described in Chapter 4 has 3 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 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.

Exercise 6.8. A proposal for a remodeled Sly's Convenience Store has 6 gas pumps each having their own waiting line. The inter-arrival time distribution of the arriving cars is Poisson with a mean of 1.25 per minute. 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 3 cars (1 in service and 2 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, the average number of customers waiting in each line, the overall line, and in 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.

6. Modeling Systems with Advanced Process Concepts

- 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 (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 (b).
-

Exercise 6.9. Apply the concepts of generic station modeling to the system described in Exercise 5.18. Be sure to show that your results are essentially the same with and without generic modeling.

Exercise 6.10. Suppose the shuttle bus system described in Section 6.6.3 now has 3 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 that 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.

Exercise 6.11. Reconsider the STEM Career Fair Mixer Example 6.1. The results indicated that the utilization of the JHBunt recruiters was still high, around 90%, and the MalWart recruiters was low, around 50% after making the resource schedule changes. Design a schedule that gets both recruiting stations to about 80% during the peak period.

Exercise 6.12. This problem examines a single item, multi-stage, 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.

Figure 6.89 illustrates a production flow line consisting of kanban based work centers.

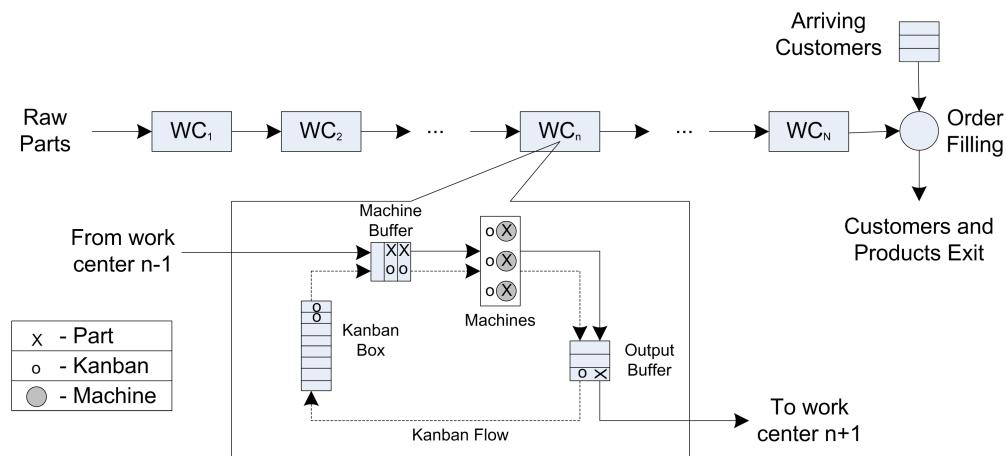


Figure 6.89.: Kanban workcenter

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

6. Modeling Systems with Advanced Process Concepts

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 are 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's assume that each machine's service time distribution is an exponential distribution with service rate μ_n . Demands from customers arrive to the system according to a Poisson process with rate λ . 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
μ_1	μ_2	μ_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 15000 hours with a warm up period of 5000 hours. Suppose $C_q = 5$, what is the effect on the system.

7. Modeling Systems with 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

Chapter 4 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 4, 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

7. Modeling Systems with Entity Movement and Material Handling Constructs

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 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 7.2 presents how facilitates resource constrained movement using the TRANSPORTER module and its accompanying constructs. A transporter is a special kind of resource in that can move. Since not all movement is as freely moving through space as people walking or fork trucks moving, provides constructs for modeling entity movement when the space between the locations becomes an important aspect of the modeling. Section 7.3 indicates how conveyors can be represented within and how they represent the space between stations. Then, in Section 7.4, 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, etc.) As usual, these concepts will be illustrated with example models.

7.1. Resource Constrained Transfer

When an entity requires the 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's revisit the test and repair shop from Chapter 4.

Recall that in the test and repair shop, parts follow 1 of 4 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 4, 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 2-4 minutes uniformly distributed. The diagnostic station had two diagnostic machines and each test station had 1 testing machine. Finally, the repair station had 3 workers that 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 (1 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.

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 2 workers at the diagnostic station, 1 worker per testing station, and 3 workers at the repair station. Thus, there are a total of 8 workers in the system. For simplicity, assume that any of these 8 workers are capable of moving parts between the stations. For example, when a part completes its operation at the diagnostic station, any worker

7.1. Resource Constrained Transfer

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 7.1 illustrates the new situation for the test and repair shop involving the use of workers.

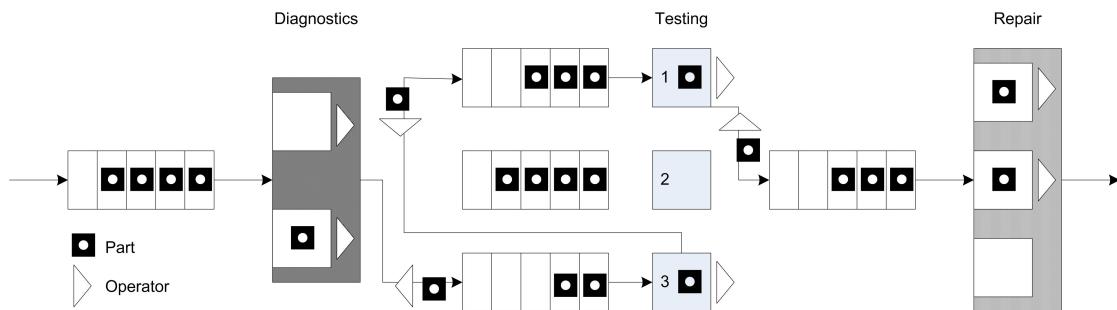


Figure 7.1.: Test and repair shop with workers providing the movement

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 7.2 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 that 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 cur-

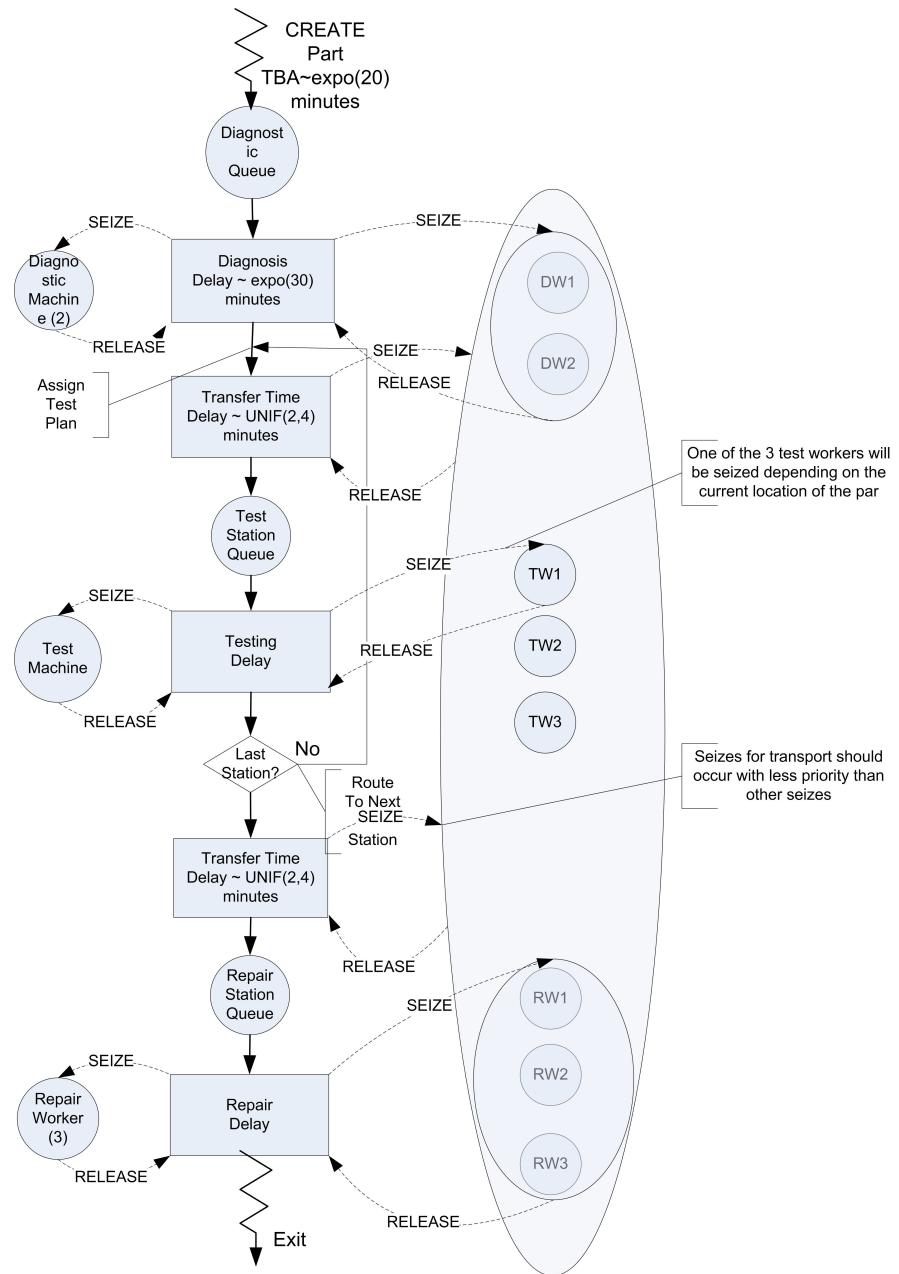


Figure 7.2.: Activity diagram for revised test and repair situation

7.1. Resource Constrained Transfer

rent station. Finally, at the repair station, the part will seize one of the repair workers. In order to receive transport, the part will seize from the entire worker set.

7.1.1. Implementing Resource Constrained Transfer

Based on Figure 7.2, the basic pseudo-code for resource constrained transfer should be something like that shown in the following pseudo-code.

```
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
        ROUTE for transfer time by sequence to STATION Test
    ELSE
        ROUTE for transfer time by sequence to STATION Repair
    ENDIF
END DECIDE

STATION Repair
RELEASE worker
SEIZE repair worker from repair worker set
DELAY for repair time
RELEASE repair worker
RECORD statistics
DISPOSE
```

7. Modeling Systems with Entity Movement and Material Handling Constructs

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.

After arriving at the test station, the part releases the worker that 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 modules, 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, transporter), and the routing of the entity. Now let's take a look at how to modify the test and repair model from Chapter 4 to use these new constructs.

Starting with a copy of the *RepairShop.doe* file from Chapter 4, 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 7.3, where the workers have been added 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.

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 7.3.: Resources for test and repair shop

Use the SET module on the Basic Process panel to define each of the three sets required for this problem. Figure 7.4 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 just listed them as shown. Recall that for the preferred resource selection rule, the order of the resources matters. The cyclical rule is used in this model.

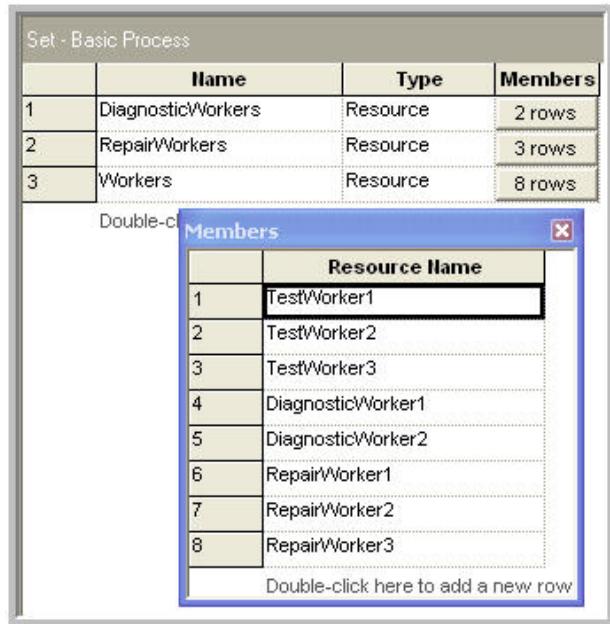


Figure 7.4.: Resource sets for test and repair shop

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 released. This is illustrated in Figure 7.5. 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 of 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 fashion, 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 3 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's start with specifying how to leave the diagnostic station. Delete the ROUTE module associated with

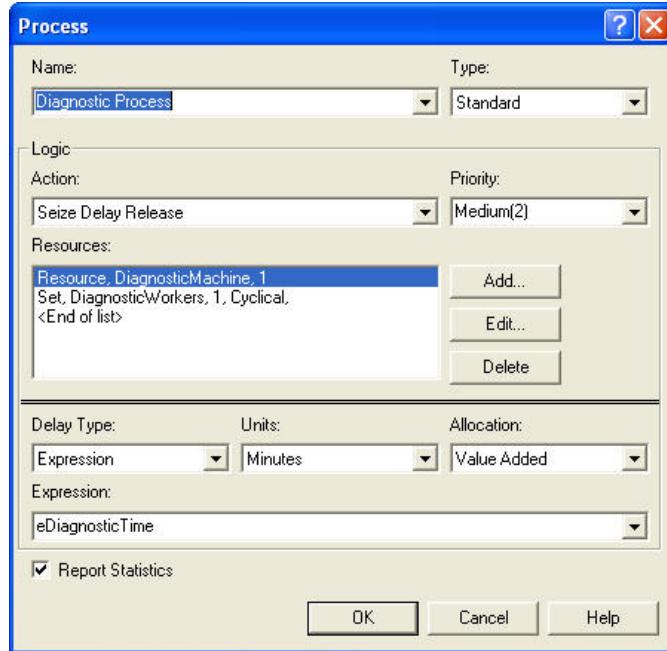


Figure 7.5.: Seizing with the diagnostic process

the diagnostic station and drag a LEAVE module into its place. Then fill out the LEAVE module as shown in Figure 7.6.

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 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

7.1. Resource Constrained Transfer

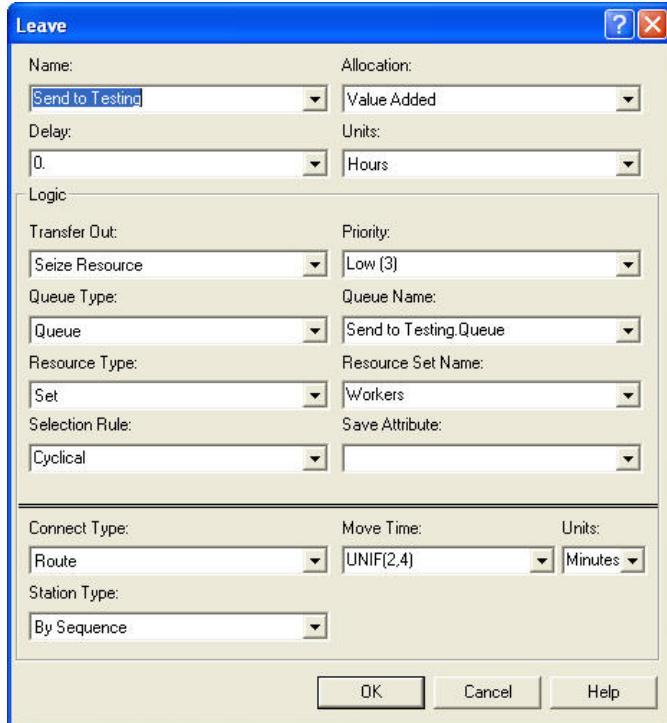


Figure 7.6.: LEAVE module for test and repair shop

be replaced with LEAVE modules like that shown in Figure 7.6. To do this replacement quicker, you can simply delete the ROUTE modules and copy/paste LEAVE modules as per Figure 7.6. You should make sure to change the name of the modules after the cut/paste operation, since no two modules can have the same name.

Now, the ENTER module can be specified. The ENTER module for test station 1 is given in Figure 7.7. 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 that 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. 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 4, the model was run for 10 replications of 4160 hours to examine the effect on the performance measures. As can be seen in Figure 7.8, the probability

7. Modeling Systems with Entity Movement and Material Handling Constructs

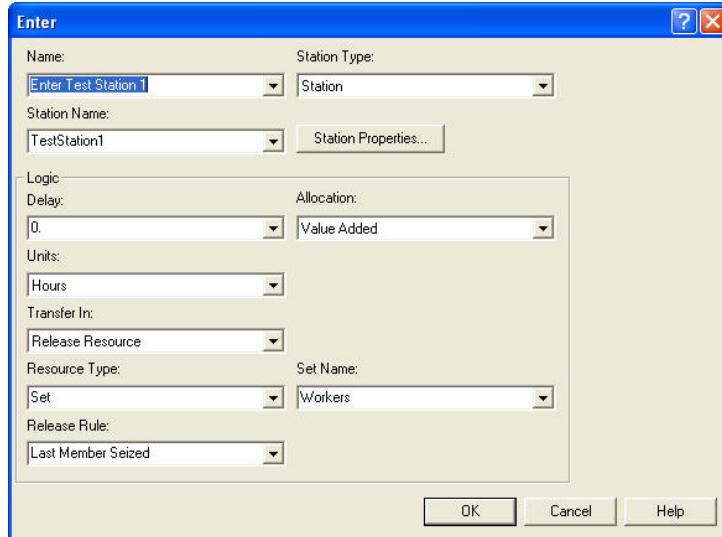


Figure 7.7.: ENTER module for test and repair shop

of meeting the contract specification has been reduced by about 10% (from 82.51% in Chapter 4 to 72.04% in this example). This is due to the increase in the average system time.

User Specified		
Tally		
Expression	Average	Half Width
ProbLTContractLimit	0.7204	0.04
SystemTimeStat	408.51	25.74

Figure 7.8.: Probability of meeting the contract requirements

As can be seen in Figure 7.9, the time spent waiting in the station's output buffer for pick up ranges from 13-15 minutes. Since there are 4 queues, this time has added significantly to the system time of the parts. In Figure 7.10, 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.

7.1. Resource Constrained Transfer

Time		
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
Other		
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 7.9.: Queue statistics for test and repair shop 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 7.10.: Utilization statistics for test and repair shop with resource constrained transfer

7.1.2. 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 for the ROUTE modules. Unfortunately, you cannot see the entities during the transfer between stations. In this section, you will augment the basic flow chart 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 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 and will have a rather crude representation of the resource usage.

Figure 7.11 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. 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.

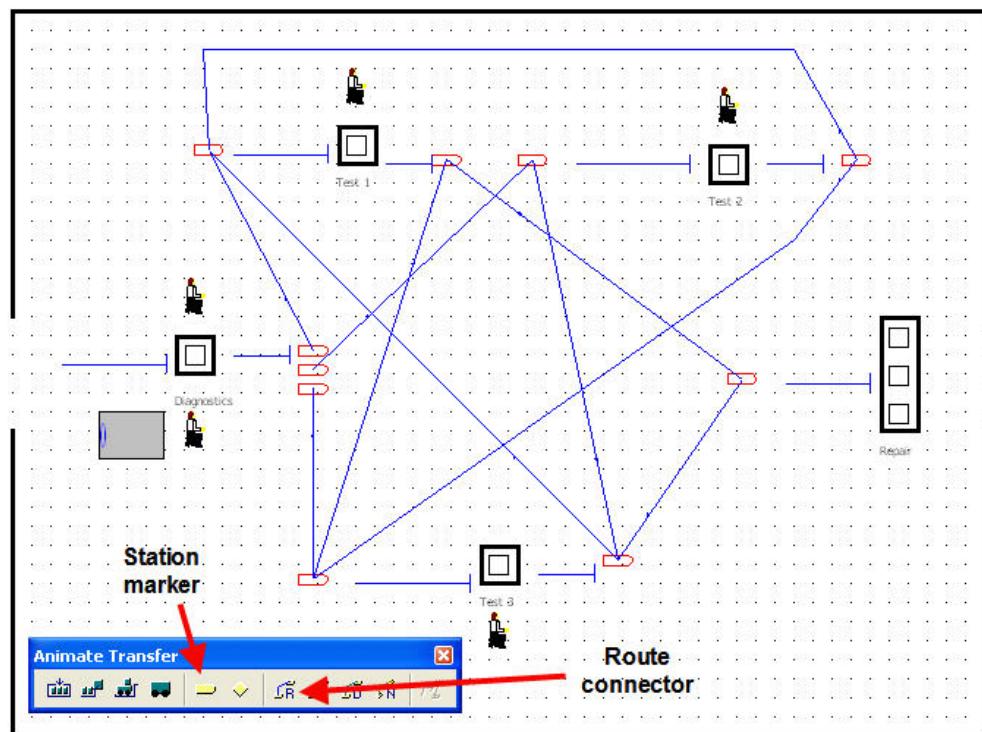


Figure 7.11.: Final animation for resource constrained transfer

7.1. Resource Constrained Transfer

An overview of how to create this animation starting from the file *RepairShopResourceConstrainedTransfer.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 2) was used to assign a seated worker to the idle picture and a worker that looks to be walking to the busy state of the diagnostic and test workers.
3. Delete the queues from the flow chart area and re-add 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 2.
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.

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 7.1.

Table 7.1.: Distribution of Test Plans

Test Plan	% of parts	Sequence
1	25%	2,3,2,1
2	12.5%	3,1
3	37%	1,3,1
4	25%	2,3

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 7.2, "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.

Table 7.2.: From/To Indicator Table

Station	Test 1	Test 2	Test 3	Repair
Diagnostics	Yes	Yes	Yes	No
Test 1	No	No	Yes	Yes
Test 2	Yes	No	Yes	No
Test 3	Yes	Yes	No	Yes

7. Modeling Systems with Entity Movement and Material Handling Constructs

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 7.12, you can place a station marker for coming into the station and for 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 7.2.

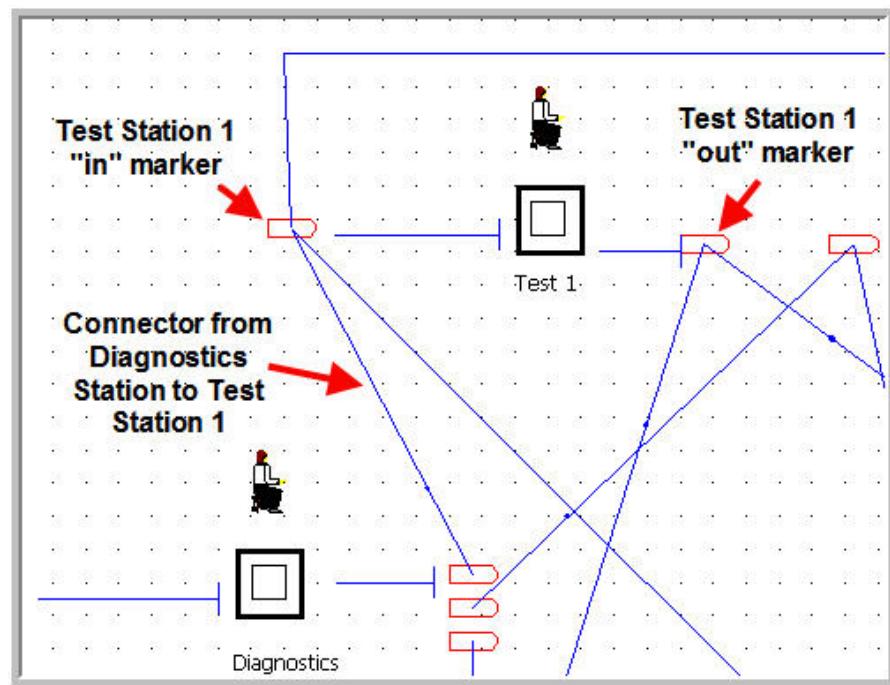


Figure 7.12.: Station markers and route connections

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 etc. 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 be-

come 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 the STATESET module would need to be used for the resources. You should consult the 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.

7.2. 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:

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 pre-defined 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 is of key interest within the modeling.

To model with transporters, 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.

TRANSPORT 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 possession of the transporter unit. Before the transporter can be reallocated to another requesting entity the transporter must be de-allocated using the FREE module.

FREE Causes the entity to de-allocate 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 can not 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 e.g. number of transporter units, default velocity, initial location, etc. 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 non-negative 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 (either when the transporter is 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 7.13 illustrates the general case of using a transporter and how it relates to the 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 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.

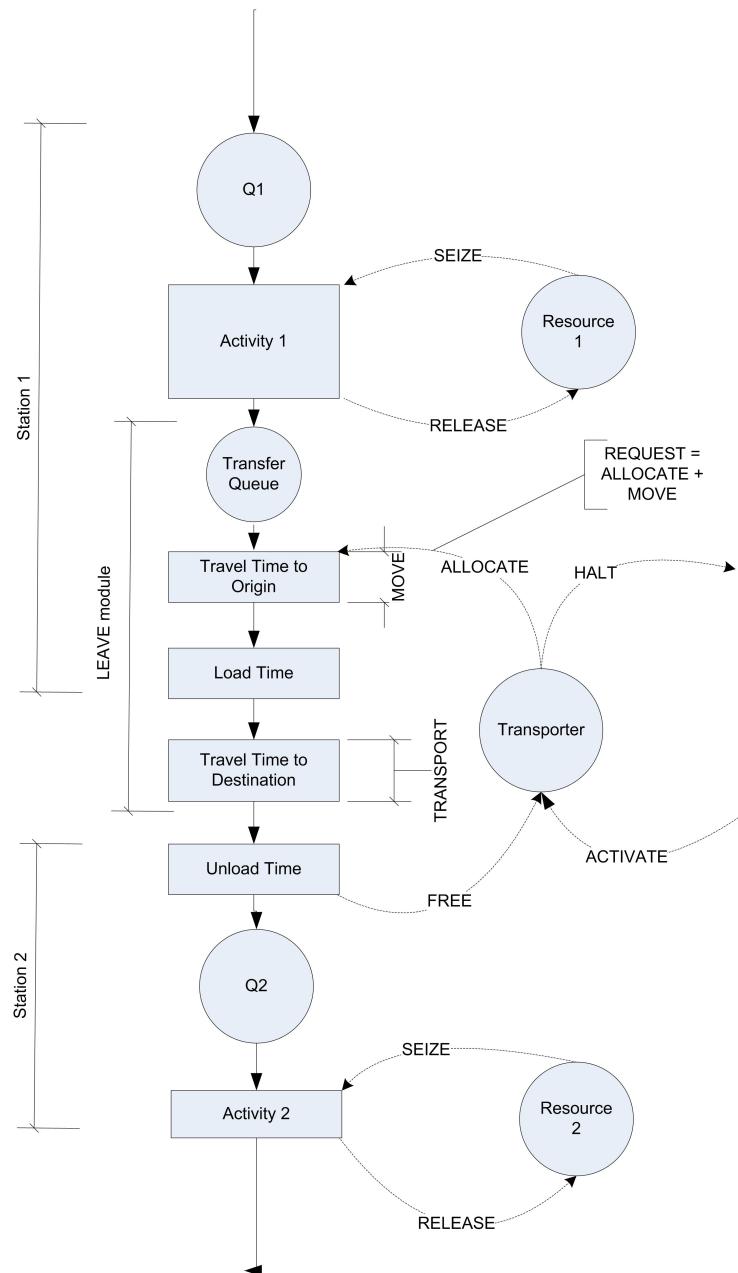


Figure 7.13.: Activity diagram for general transporter case

7. Modeling Systems with Entity Movement and Material Handling Constructs

The activity diagram in Figure 7.13 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.

7.2.1. 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 `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 7.3. 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 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).

Table 7.3.: Transporter distances between stations

Station	Diagnostics	Test 1	Test 2	Test 3	Repair
Diagnostics	—	40	70	90	100
Test 1	43	—	10	60	80
Test 2	70	15	—	65	20
Test 3	90	80	60	—	25
Repair	110	85	25	30	—

7.2. Constrained Transfer with Transporters

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 7.14 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.

DISTANCE module data (Stations table):

	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 table:

	Name	Stations
1	TransporterDistances	20 rows

Figure 7.14.: DISTANCE module for general transporter case

The TRANSPORTER module, see Figure 7.15, 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 3 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

7. Modeling Systems with Entity Movement and Material Handling Constructs

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.

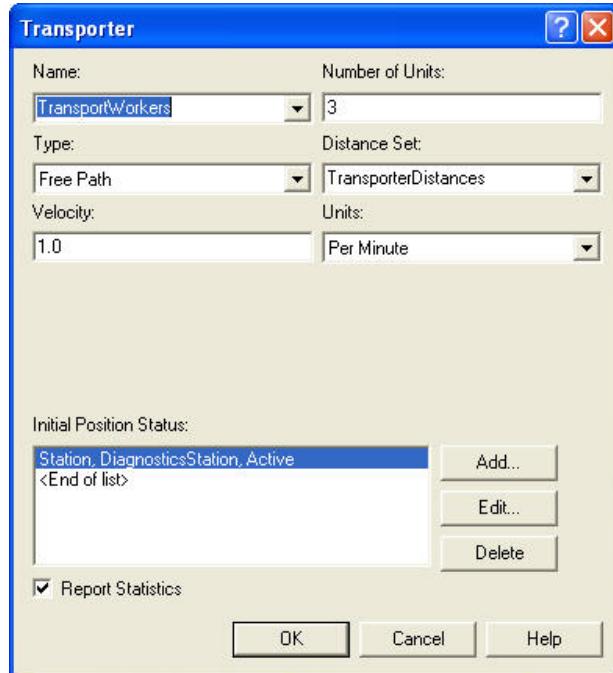


Figure 7.15.: TRANSPORTER module for general transporter case

Now that the data modules are defined, let's take a look at the overall model. Figure 7.16 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 straight forward. You just need to specify that the type of transfer in option is Free Transporter and indicate what transporter to free. Figure 7.17 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 7.18 shows the REQUEST module for the example. In the REQUEST module, you must

7.2. Constrained Transfer with Transporters

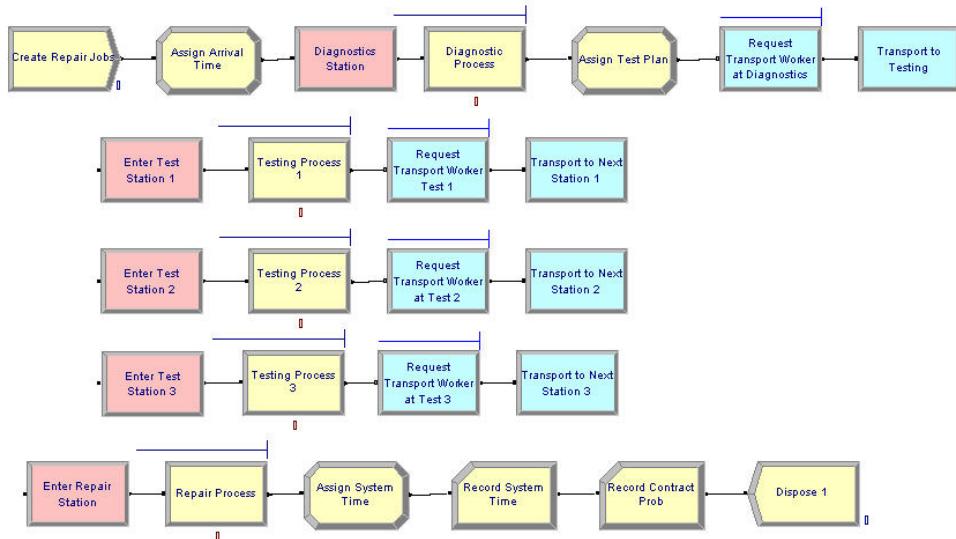


Figure 7.16.: Overall test and repair model with transporters

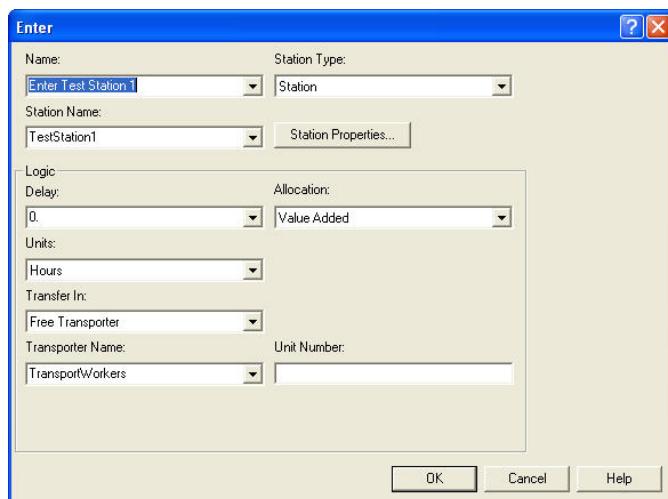


Figure 7.17.: ENTER module with free transporter option

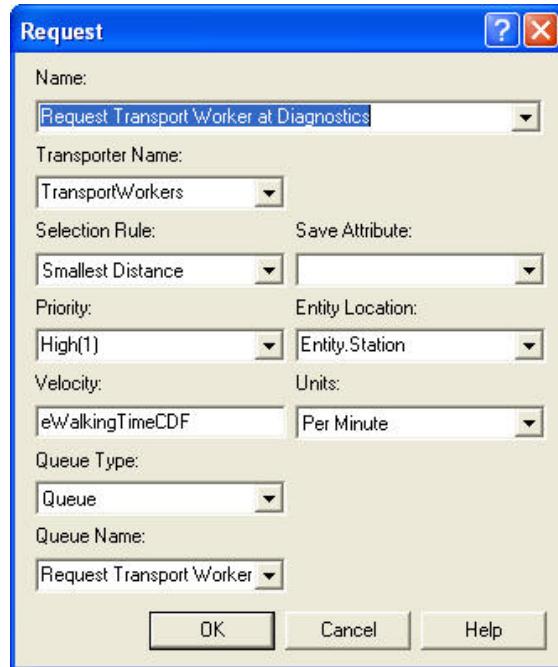


Figure 7.18.: REQUEST module

specify which transporter to request and give the selection rule. The selection rules are the 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, 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.

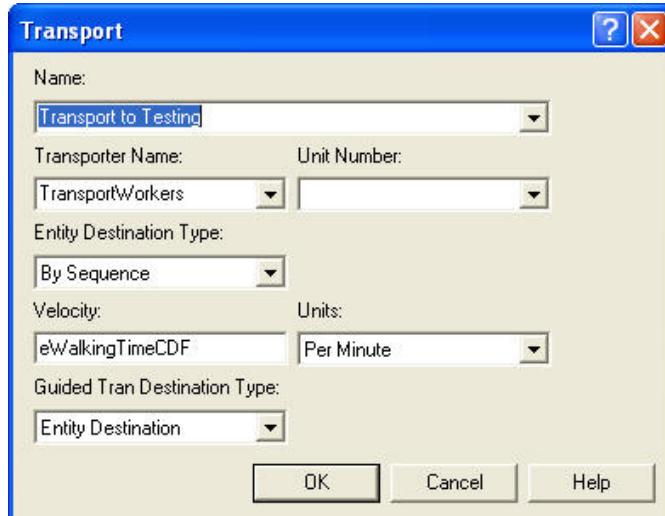


Figure 7.19.: TRANSPORT module

Figure 7.19 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

7. Modeling Systems with Entity Movement and Material Handling Constructs

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 different than 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 7.15 that 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, then the number busy and utilization of the transporters will be reported automatically.

Transporter		
Usage		
Number Busy	Average	Half Width
TransportWorkers	0.4660	0.00
Number Scheduled		
Number Scheduled	Average	Half Width
TransportWorkers	3.0000	0.00
Utilization		
Utilization	Average	Half Width
TransportWorkers	0.1553	0.00

Figure 7.20.: Transporter statistics

As can be seen in Figure 7.20, the utilization of the three transporters is very low. Less than three workers probably need for the transport task. The reader is asked to explore this issue an exercise.

7.2.2. 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 flow chart 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 7.21. 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).
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 ($NT(\text{TransportWorkers})$). 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 7.22.

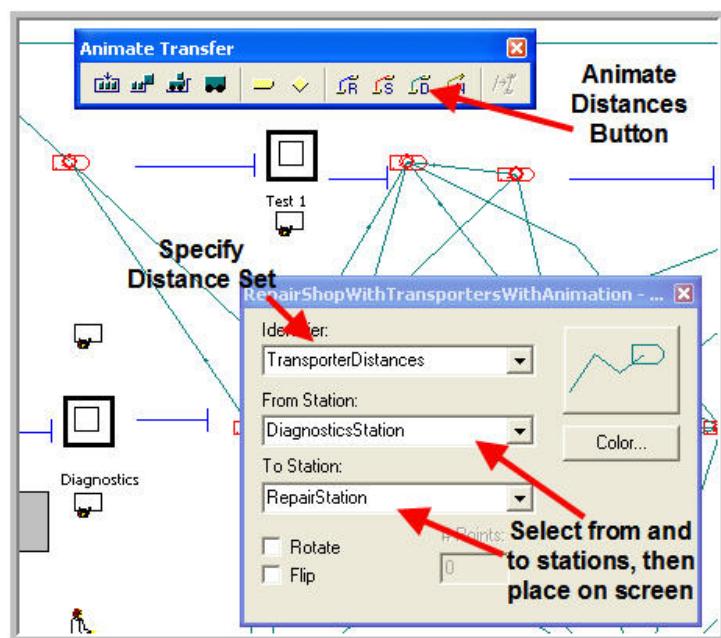


Figure 7.21.: Placing the distance animation elements

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 was used that showed a person sitting for idle and walking/carrying something to show busy. Another way to show that

7. Modeling Systems with Entity Movement and Material Handling Constructs

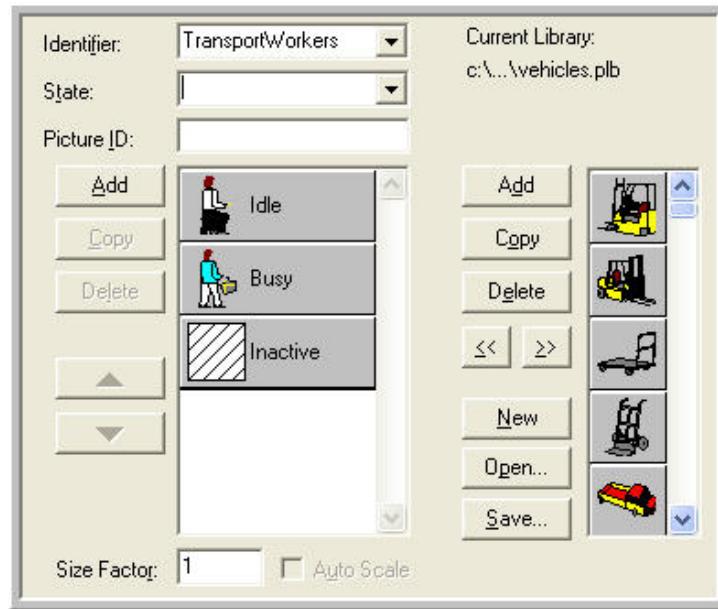


Figure 7.22.: Defining the transporter picture

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 to not use the ENTER module. The basic idea would be implemented as follows:

```

STATION
DECIDE
    IF requests are waiting
        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, e.g. 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.

7.3. 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 20-80 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.

Non-accumulating 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 non-accumulating conveyor (the steps are like a bucket for the person to ride in. People movers in airports are like non-accumulating, 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

7. Modeling Systems with Entity Movement and Material Handling Constructs

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) discuss 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, if there actually 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 , a conveyor is a material-handling device for transferring or moving entities along a pre-determined path having fixed pre-defined 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 7.23 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.

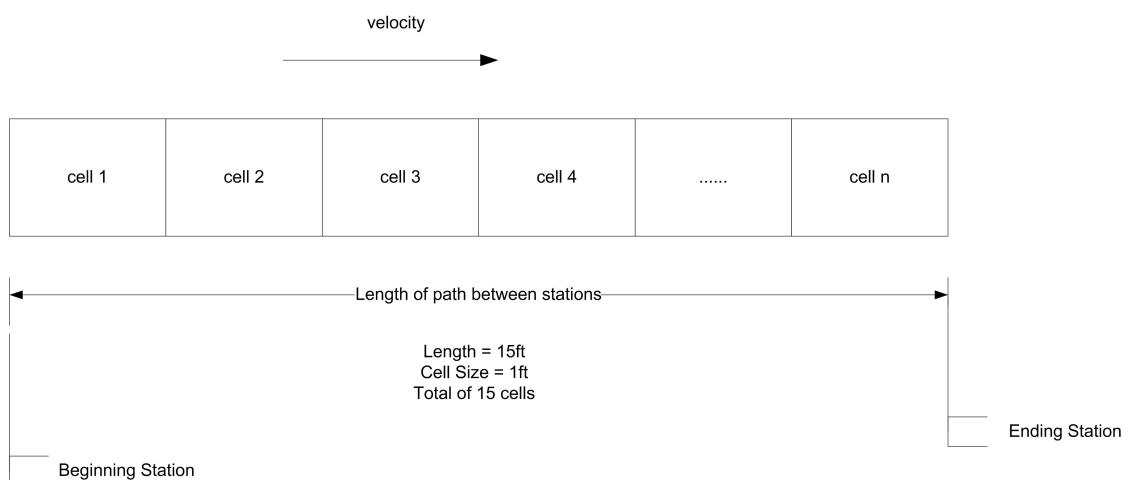


Figure 7.23.: A conveyor conceptualized as a set of contiguous cells

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 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 key modules for modeling the use of conveyors are:

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 non-accumulating 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

7. Modeling Systems with Entity Movement and Material Handling Constructs

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 differently for accumulating and non-accumulating conveyors.

Non-accumulating 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 7.24, the circle needed 1 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 1 cell at a time.

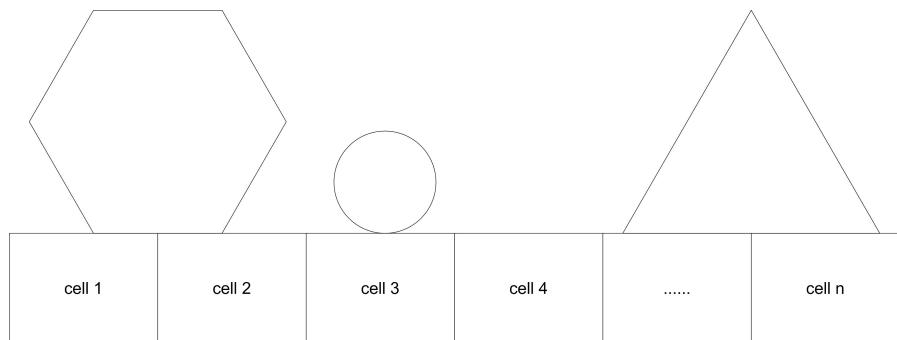


Figure 7.24.: Different entity sizes on a conveyor

Figure 7.25 illustrates an approximate activity flow diagram for an entity using a non-accumulating 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 foot 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 3rd cell and then releases the first cell before delaying for the move time for the 3rd 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).

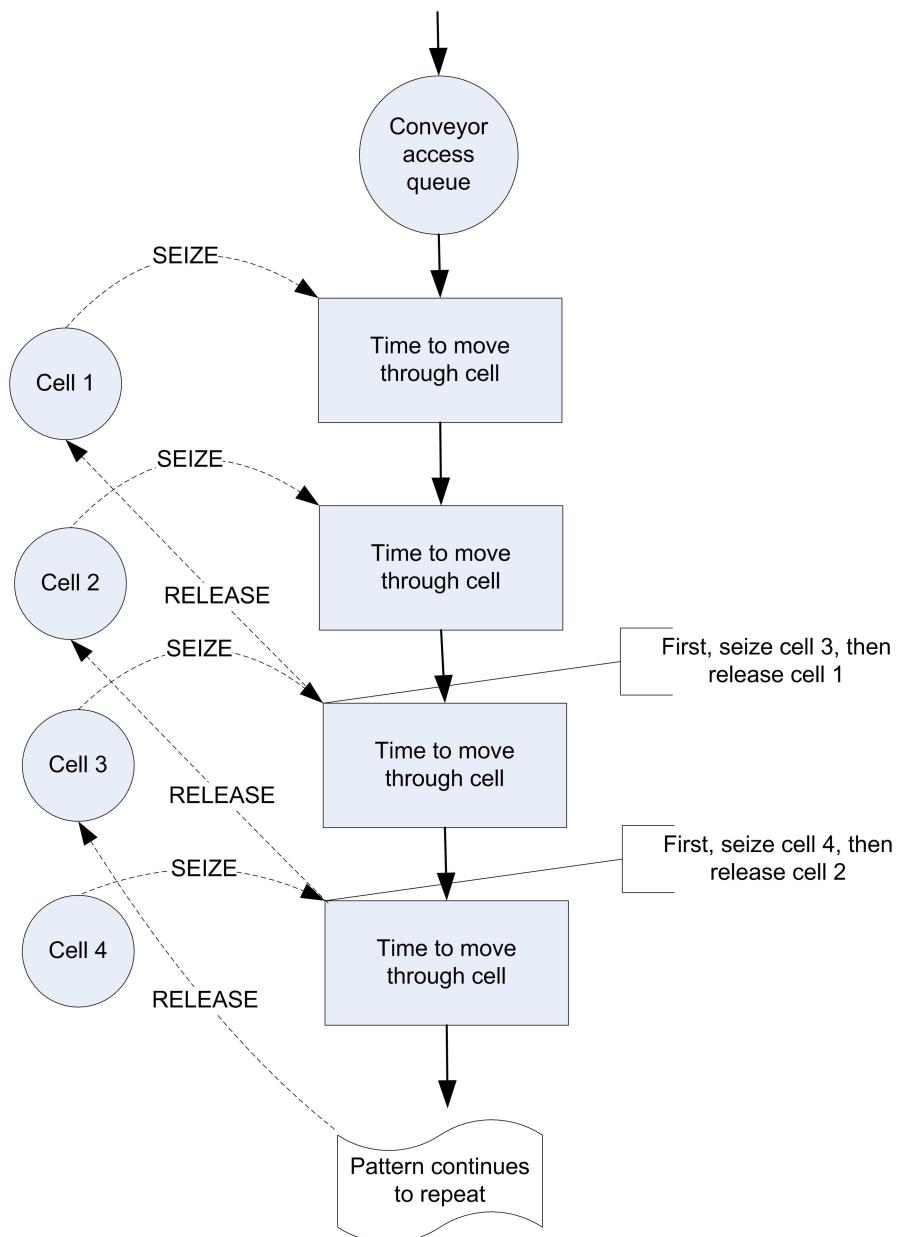


Figure 7.25.: Activity diagram for non-accumulating conveyor

7. Modeling Systems with Entity Movement and Material Handling Constructs

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 fashion. 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 got a whole cell of 1 foot before it moved onto the conveyor.

Now you are ready to put these concepts into action.

7.3.1. Test and Repair Shop with Conveyors

For simplicity, the test and repair shop will be used for illustrating the use of conveyors. Figure 7.26 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

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.

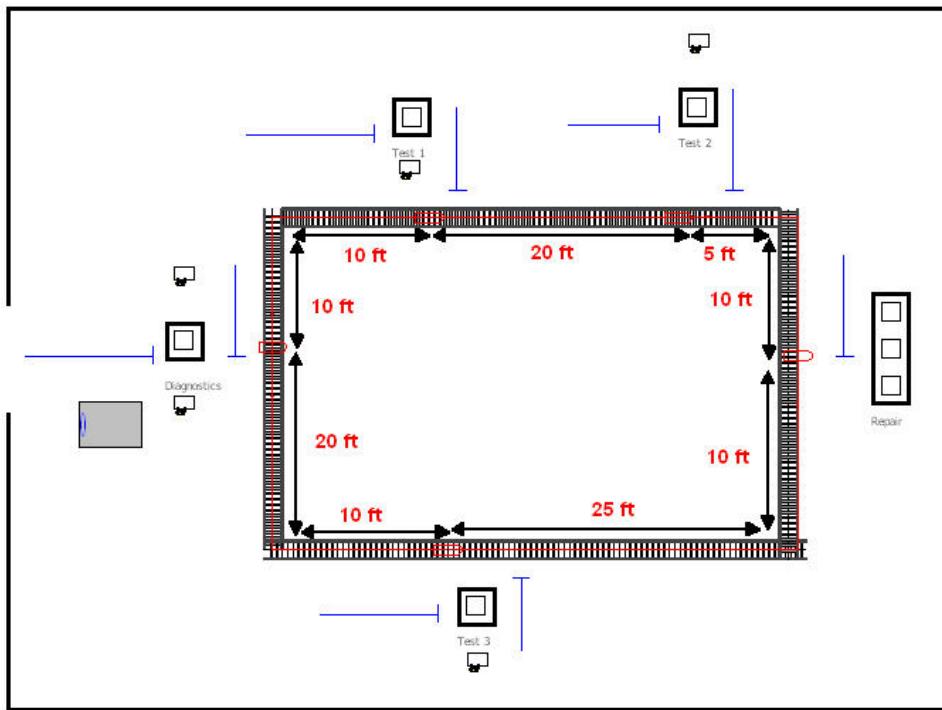


Figure 7.26.: Test and repair shop with conveyors

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 7.27.
4. Save your file. The completed model can be found in the file *RepairShopWithConveyorsWith-NoAnimation.doe*.

Variable - Basic Process																	
	Name	Rows	Columns	Data Type	Clear Option	Initial Values	Report Statistics										
1	vContractLimit					rows	<input type="checkbox"/>										
2	vMTBA					rows	<input type="checkbox"/>										
3	vSizes	4				rows	<input type="checkbox"/>										
Double-click here to add a new row.																	
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th colspan="2">Initial Values</th> </tr> <tr> <td>1</td><td>1</td></tr> <tr> <td>2</td><td>1</td></tr> <tr> <td>3</td><td>2</td></tr> <tr> <td>4</td><td>2</td></tr> </table>								Initial Values		1	1	2	1	3	2	4	2
Initial Values																	
1	1																
2	1																
3	2																
4	2																

Figure 7.27.: Array for part sizes

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 7.28 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

7. Modeling Systems with Entity Movement and Material Handling Constructs

here since this is where the parts enter the system.

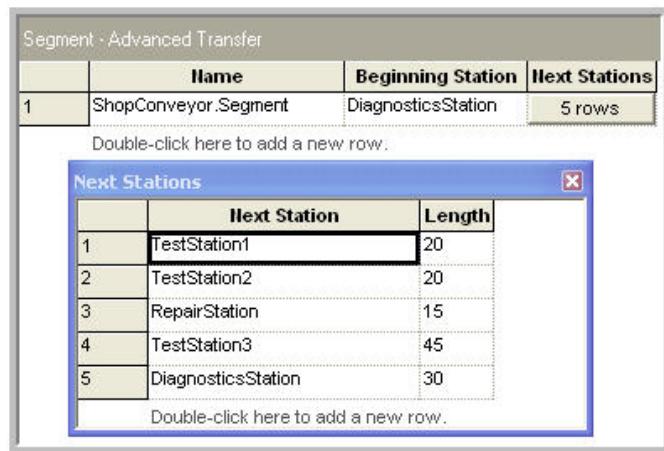


Figure 7.28.: SEGMENT module for test and repair conveyor

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 7.26. The lengths specified in Figure 7.28 are in the actual physical distances (e.g. feet). The mapping to cells occurs in the CONVEYOR module.

Figure 7.29 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 non-accumulating conveyor should be used.

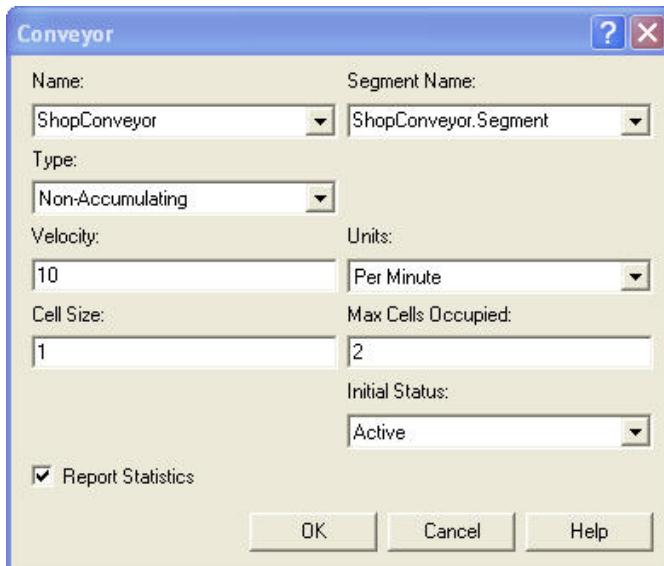


Figure 7.29.: CONVEYOR module for test and repair example

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 7.25, 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 foot 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 that 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 7.30.

Assignments			
	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 7.30.: Assigning the size of the parts

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 7.31 using the entity's `mySize` attribute. Each of the other LEAVE modules for the other stations should be updated in a similar fashion. Now, you must update the ENTER module so that the entity can exit the conveyor after arriving to the desired station.

In Figure 7.32, the Exit Conveyor option is used when transferring into the Test 1 Station. In both of 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. See Figure 7.33. 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

7. Modeling Systems with Entity Movement and Material Handling Constructs

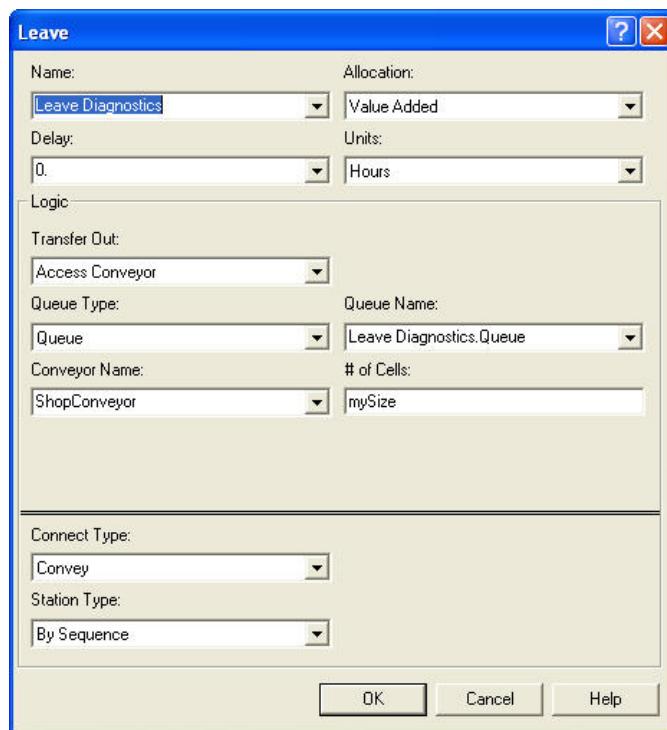


Figure 7.31.: LEAVE module for test and repair example

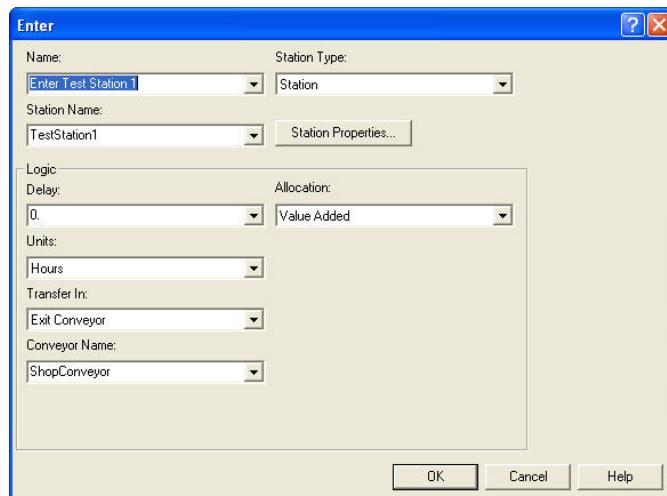


Figure 7.32.: ENTER module for test station 1

statistics for the conveyor 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.

Conveyor		
Usage		
Blocked	Average	Half Width
ShopConveyor	0.00	0.00
Utilization	Average	Half Width
ShopConveyor	0.00904537	0.00
User Specified		
Tally		
Expression	Average	Half Width
ProbLTContractLimit	0.8093	0.02
SystemTimeStat	361.04	8.75

Figure 7.33.: Statistics for test and repair shop with conveyors

7.3.2. 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 set, with appropriately drawn pictures for each type of plan. This is illustrated in Figure 7.34 and Figure 7.35. The assignment to the Entity.Picture attribute is shown in Figure 7.39.

To update the previous transporter animation, perform the following steps.

1. Copy and paste the animation elements from the file, *RepairShopWithTransportersWithAnimation.doe* to your own model file. The completed 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 7.26. The segment animation dialog is shown in Figure 7.36. It works very much like the route and distance animation dialog.

7. Modeling Systems with Entity Movement and Material Handling Constructs

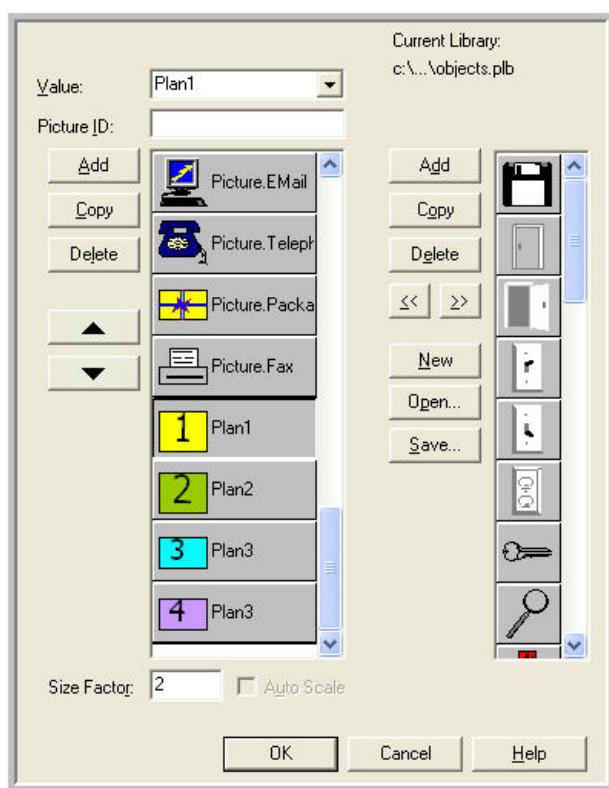


Figure 7.34.: Entity pictures for test plans

Set - Basic Process

	Name	Type	Members
1	DiagnosticWorkers	Resource	2 rows
2	RepairWorkers	Resource	3 rows
3	Workers	Resource	8 rows
4	PartPictures	Entity Picture	4 rows

Double-click here to add a new row.

Members

	Picture Name
1	Plan1
2	Plan2
3	Plan3
4	Plan4

Double-click here to add a new row.

Figure 7.35.: Picture set for test plans

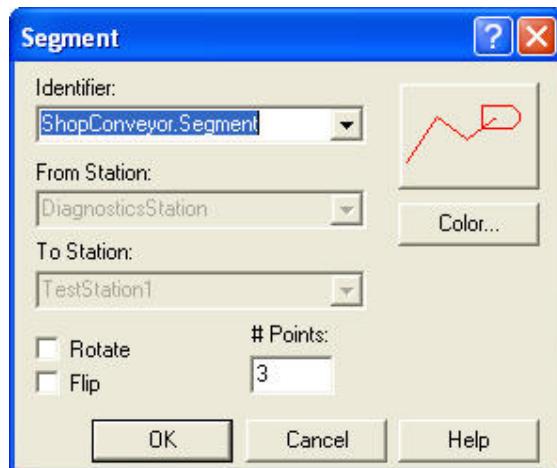


Figure 7.36.: Segment animation dialog

After the segments have been laid down, you might want to place some lines to represent the conveyors. This was done in Figure 7.36 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 non-accumulating conveyors. The next section describes how accumulating conveyors operate and discusses a number of other conveyor modeling issues.

7.3.3. 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 re-circulation conveyors.

7.3.3.1. 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. Don't ask how you got your roller blades through security! While on the people mover you aren't skating (you are standing still, resting, but still moving with the people mover). Up ahead of you,

7. Modeling Systems with Entity Movement and Material Handling Constructs

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.

In modeling accumulating conveyors, the main differences occur in the actions of the ACCESS and CONVEY modules. Instead of disengaging the conveyor as with non-accumulating 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 doesn't have to be divisible into an even number of cells. In addition, as can be seen in Figure 7.37, 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 non-accumulating conveyors. Thus, to illustrate accumulating conveyors, a SMART file, (*Smarts101.doe*), will be used. See Figure 7.38. The SMART files can be found within the folder within your Arena installation.

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

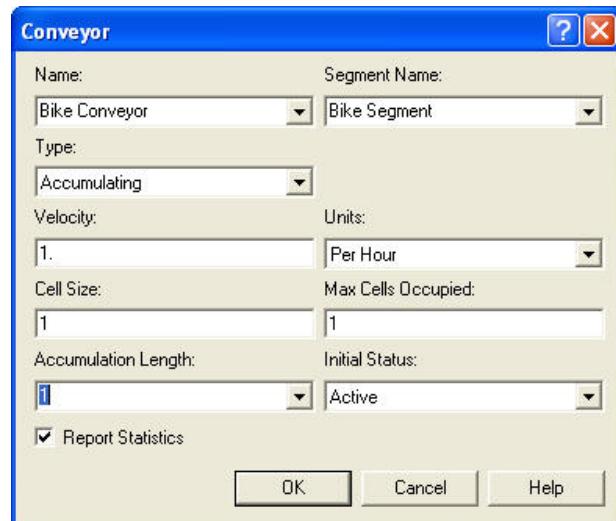


Figure 7.37.: Accumulating conveyor dialog

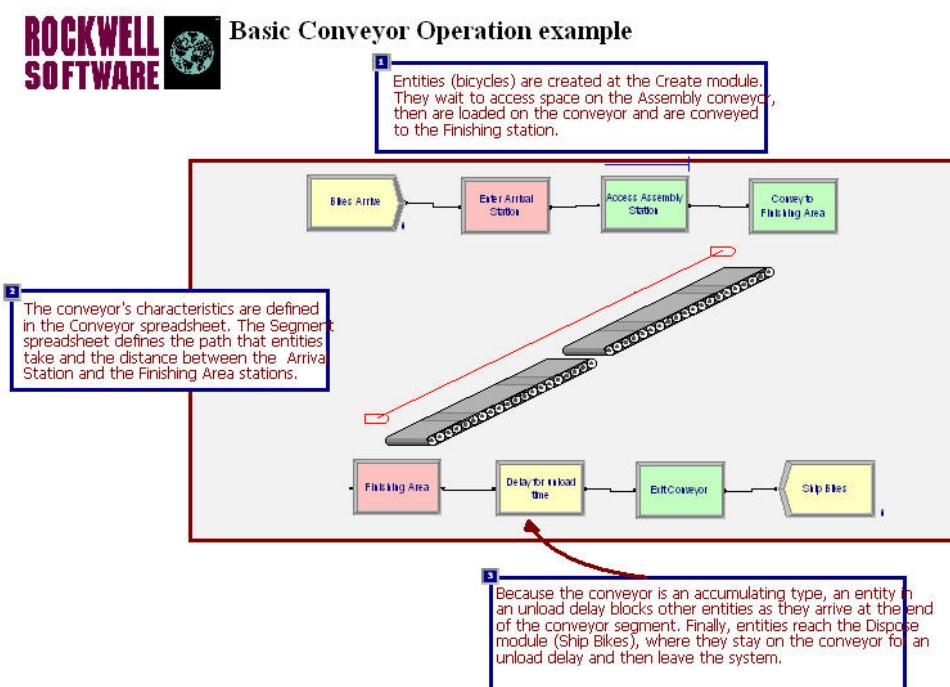


Figure 7.38.: Arena Smarts101.doe model for accumulating conveyors

7. Modeling Systems with Entity Movement and Material Handling Constructs

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 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.

7.3.3.2. 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 , 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's first take a look at *Smarts107.doe*. Figure 7.39 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 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, Figure 7.40, involves the merging of one conveyor onto another conveyor as in SMART file, *Smarts110.doe*. 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

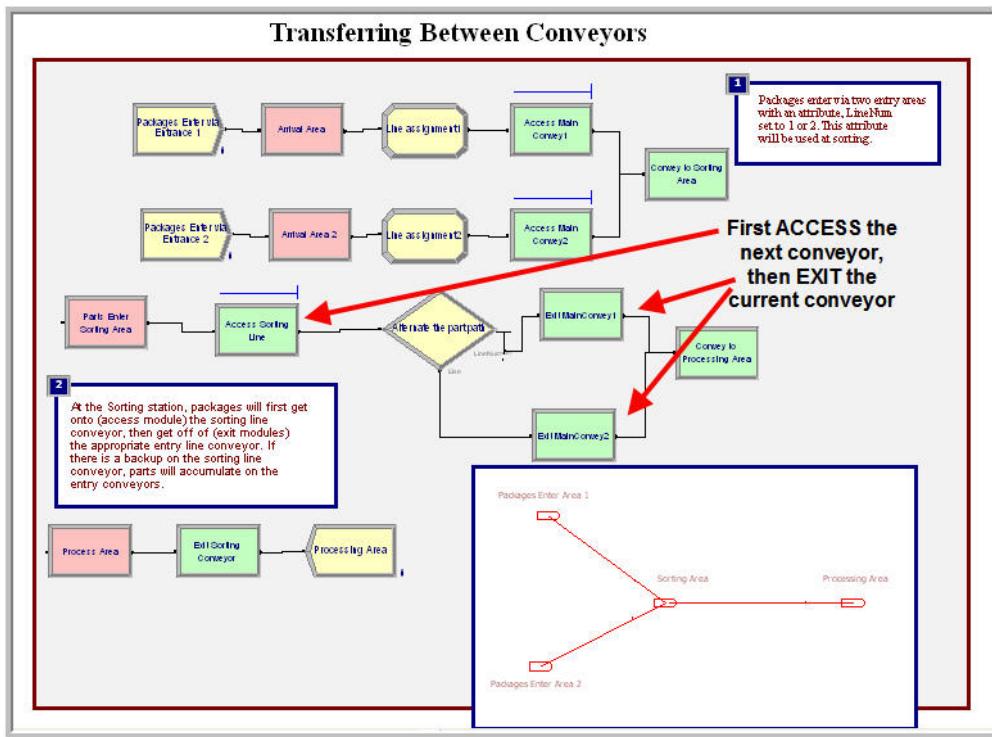


Figure 7.39.: Transferring between conveyors

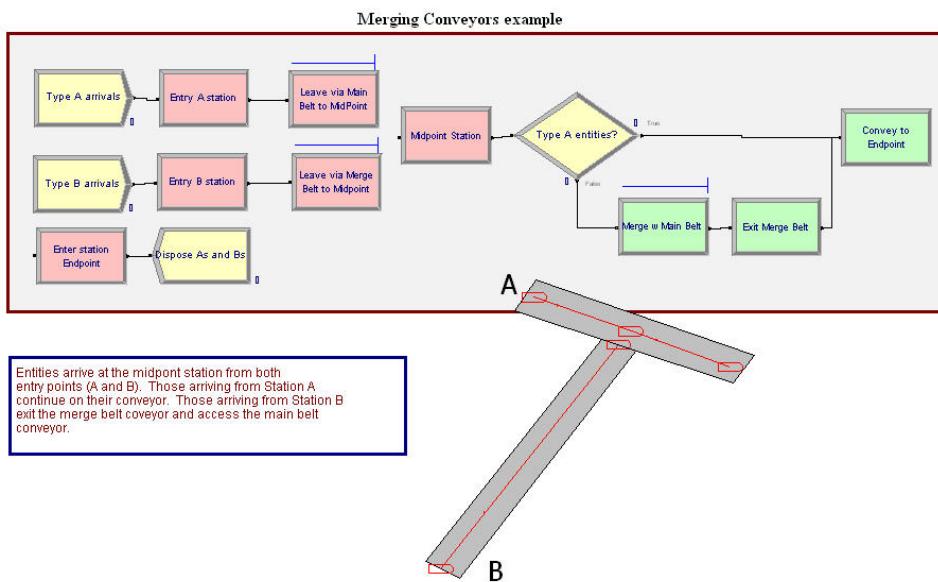


Figure 7.40.: One conveyor merging into another

7. Modeling Systems with Entity Movement and Material Handling Constructs

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.

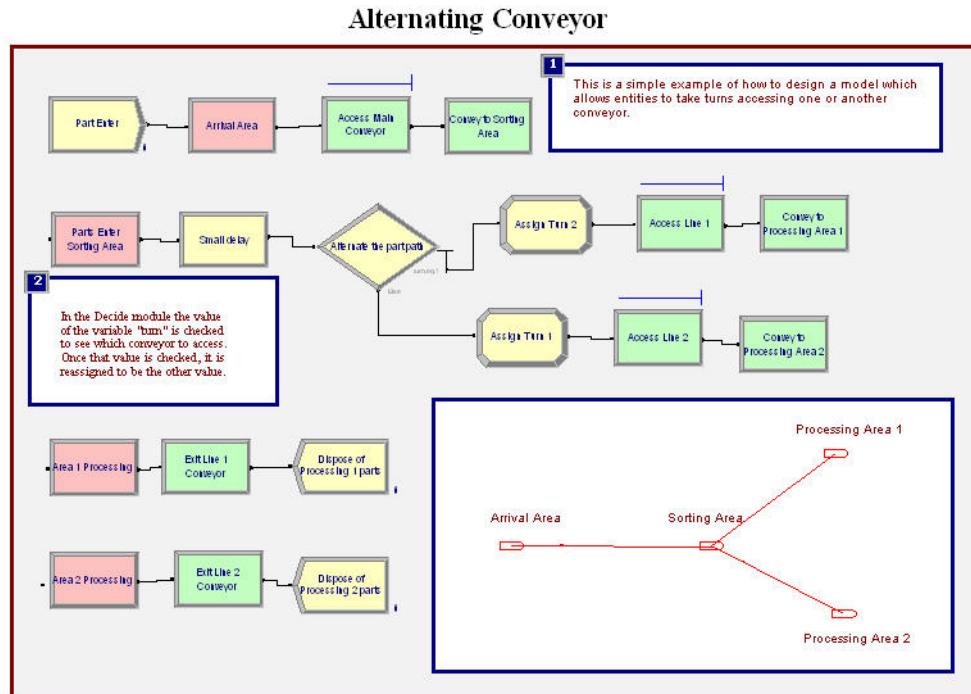


Figure 7.41.: Simple alternating conveyor

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 attributes (e.g. destination). Arena's *Smarts108.doe* file illustrates the basic concepts behind diverging conveyors. In this example, as shown in Figure 7.41, the parts arrive to the arrival area and access a conveyor to the sorting station. 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 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 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.

7.3.3.3. 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 by not exiting the conveyor while undergoing a PROCESS module.

Processing on Conveyor example

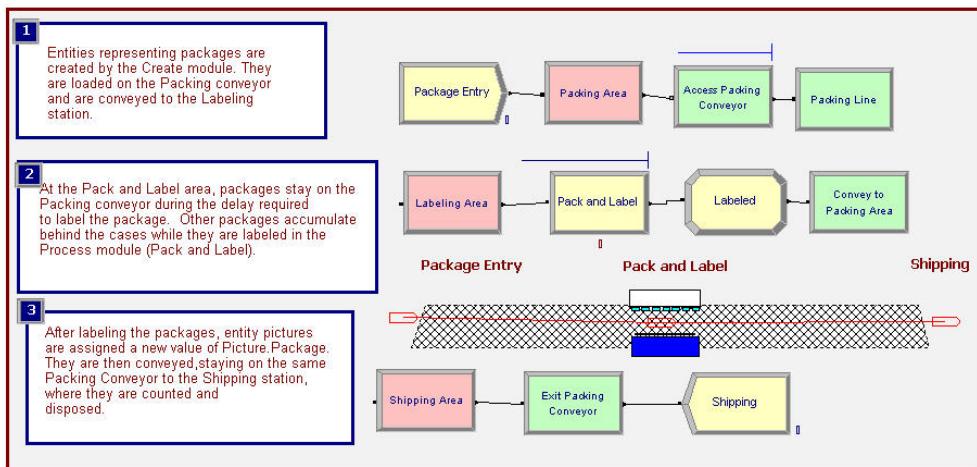


Figure 7.42.: Processing while on a conveyor

Arena's SMART file, *Smarts103.doe*, contains an example of this type of modeling. In this example, as shown in Figure 7.42, 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 non-accumulating 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.

7.3.3.4. 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 two. Thus, the size of test station two's queue 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 7.43 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 *RepairShopWithRecirculatingConveyorsWithAnimation.doe*. If you run the model and watch the animation, you will see much more parts on the conveyor because of the re-circulation.

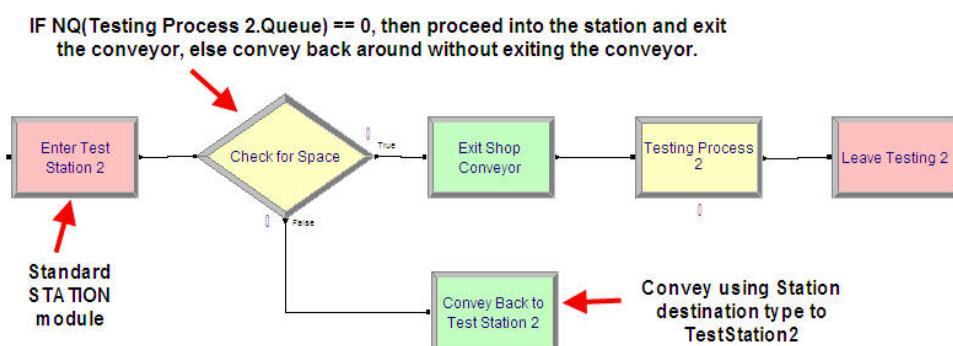


Figure 7.43.: Test and repair shop with recirculating conveyor

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. 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.

7.4. 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 be limited to AGV systems that follow a prescribed path (e.g. tape, embedded wires, etc) 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 7.44 illustrates the major concepts in guided vehicle modeling: networks, links, intersections, and stations.

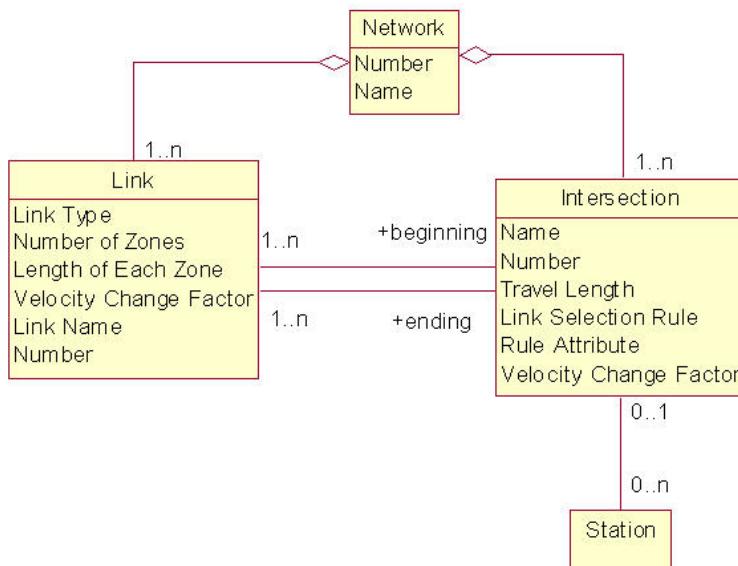


Figure 7.44.: Relational diagram for guided vehicle networks

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

7. Modeling Systems with Entity Movement and Material Handling Constructs

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 7.45 illustrates a simple three link network.

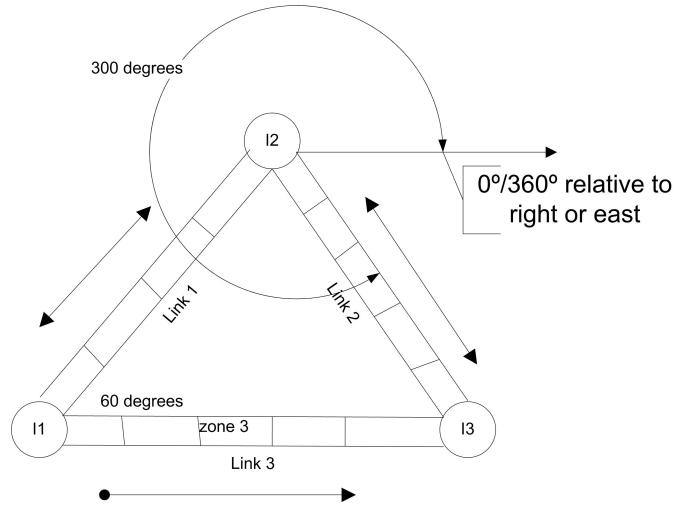


Figure 7.45.: Example three link network

In the figure, Link 1 is bidirectional with a beginning intersection label I1 and an ending intersection labeled I2. Link 1 consists of 4 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 in the figure is 60 degrees. Zero and 360 both represent to the east or right. The beginning direction for Link 2 is 300 degrees. 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 degrees 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 7.44, 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 ele-

ments that underlie guided path transporter modeling within . Clearly, a fixed path network consists of a set of links and the associated intersections. Also from Figure 7.44, 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.

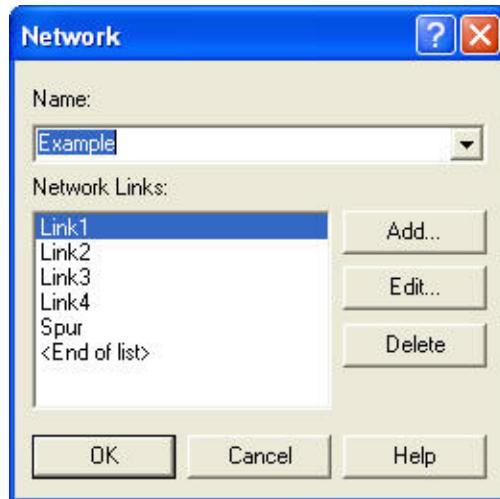


Figure 7.46.: NETWORK module for guided path modeling

The new 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 7.46 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 7.47 illustrates the NETWORK LINK module. Notice that the spreadsheet view of the module makes editing multiple links relatively easy.

Now, let's 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 the AGV (taking 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 7.48. 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's assume that the item being transported is rather large and heavy. That's 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

7. Modeling Systems with Entity Movement and Material Handling Constructs

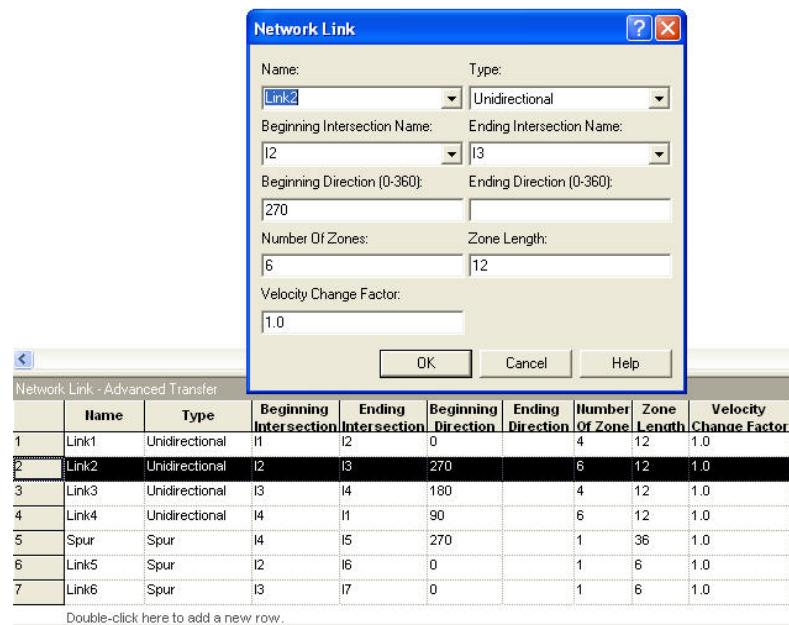


Figure 7.47.: NETWORK LINK module

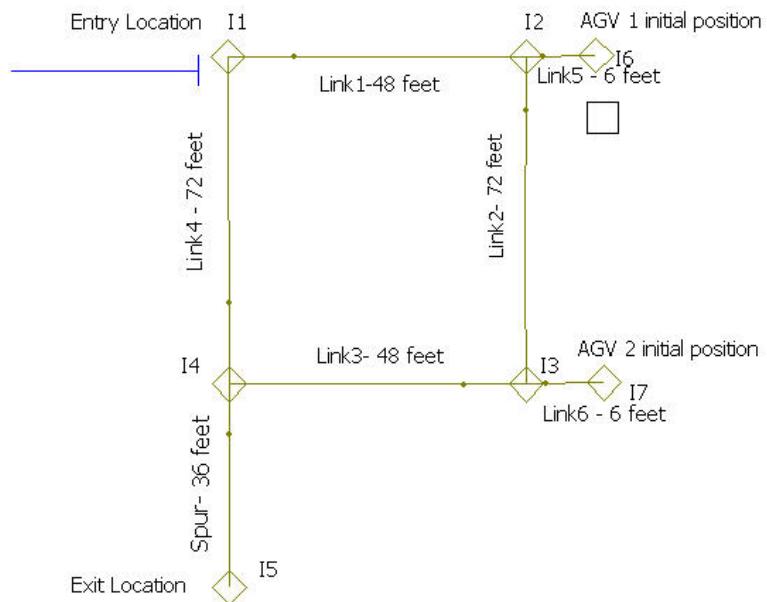


Figure 7.48.: Layout for simple AGV example

7.49 shows the TRANSPORTER module for using guided path transporters. When the “Guided” type is selected, additional options become available.

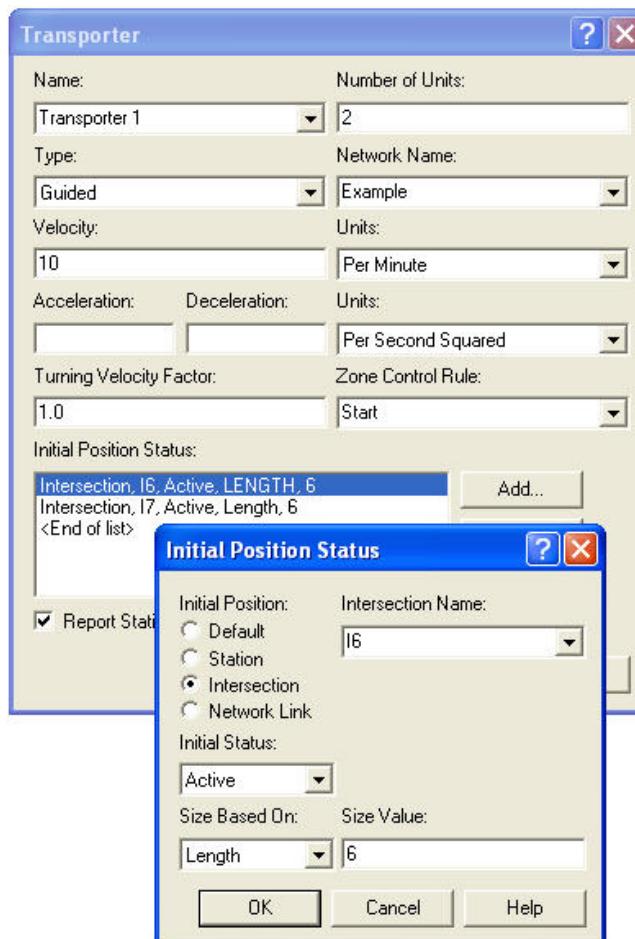


Figure 7.49.: TRANSPORTER module for simple AGV example

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 AGV's 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.

7. Modeling Systems with Entity Movement and Material Handling Constructs

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's 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 7.47, Link 1 consists of 4 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 7.47. Now, you need to decide on the direction of travel permitted on the links. In this system, let's assume that the AGVs will travel clockwise around the figure. Figure 7.47 indicates that the direction of travel from I1 along Link 1 is zero degrees (to the east). Then, Link 2 has a direction of 270, Link 3 has a direction of 180, and Link 4 has a direction of 90. The spur link to the exit station has a direction of 270 (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 : 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 7.50 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 operation of these rules, you should refer to 1 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

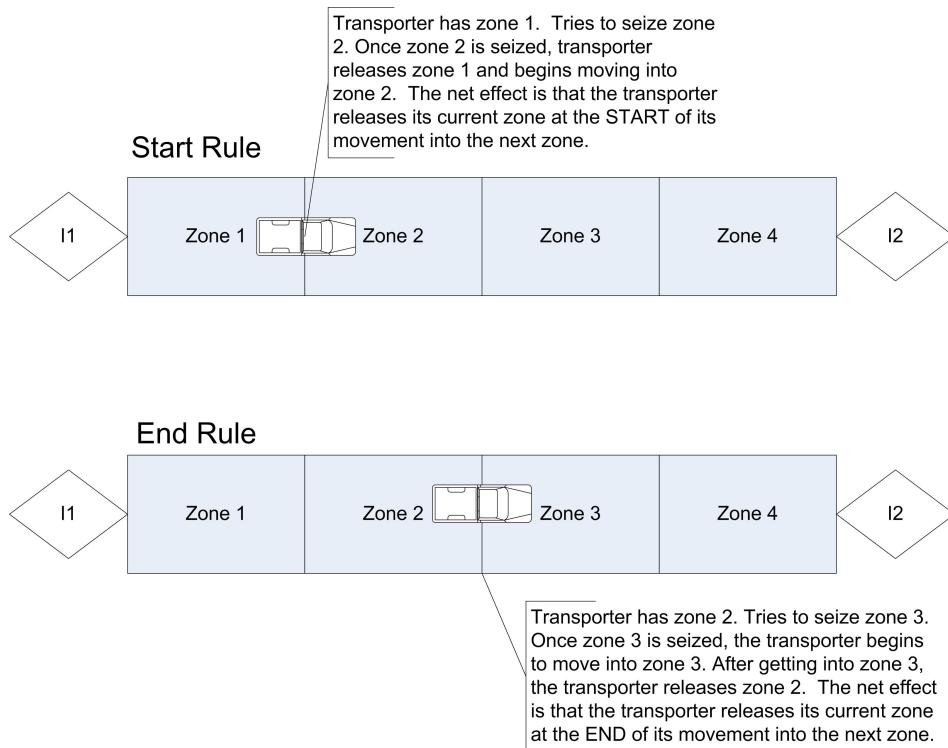


Figure 7.50.: Illustrating zone control

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, can detect when this occurs during the simulation run. If it is detected, then will stop with a run time error. In some cases, cannot detect deadlock. For example, suppose a station was attached to I4 and a transporter (τ_1) was sent directly to that station for some processing. Also, suppose that the part stays on transporter (τ_1) during the processing so that transporter (τ_1) is not released at the station. Now, suppose that transporter (τ_2) has just dropped off something at the exit point, I5, and that a new part has just come in at I1. Transporter (τ_2) will want to go through I4 on its way to I1 to pick up the new part. Transporter (τ_2) will wait until transporter (τ_1) finishes its processing at I4 and if transporter (τ_1) eventually moves there will be no problem. However, suppose transporter (τ_1) is released at I4 and no other parts arrive for processing. Transporter (τ_1) 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 (τ_1) to move, transporter (τ_2) will be stuck. There will be the one part waiting for transporter (τ_2), but transporter (τ_2) 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

7. Modeling Systems with Entity Movement and Material Handling Constructs

situations. For novice modelers, deadlock can seem troublesome, especially if you don't 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 (T_1) to receive the processing that was occurring at the station associated with I_4 , 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's briefly discuss the operation of spurs and bidirectional links. This discussion is based in part on (Pegden et al., 1995) page 394. Notice that the link from I_4 to I_5 is a spur in the previous example. A spur is a special type of link designed to help 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 7.51. 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

7.4. Modeling Guided Path Transporters

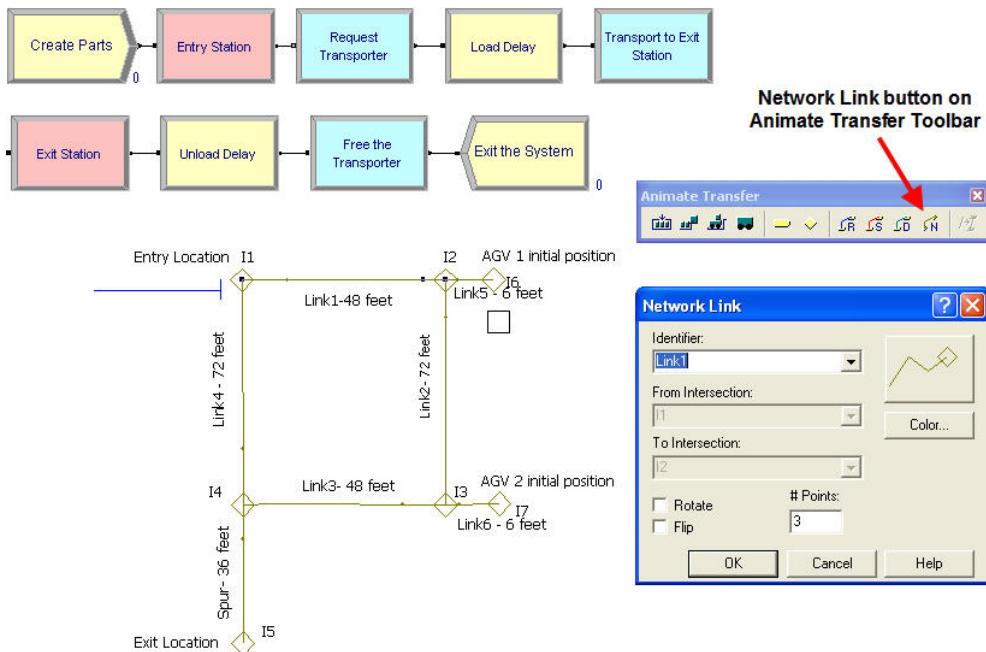


Figure 7.51.: Animating guided path transporters

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 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 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, uses standard algorithms to find the shortest path between all intersections in the network. This is used when dispatching the vehicles.
2. Specialized variables that are used in guided path networks. These variables assist with

7. Modeling Systems with Entity Movement and Material Handling Constructs

indicating where the vehicle is within the network, accessing information about the network within the simulation, (e.g. distances, beginning intersection for a link, etc.), 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

- 1.
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 1. Advanced examples are also given in both of those texts, including within 1 the possible use of guided path constructs for modeling automated storage and retrieval systems (AS/RS) and overhead cranes.

7.5. 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 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 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. Thus, in the next chapter, we will apply a full range of modeling constructs to a realistic problem situation.

7.6. Exercises

Exercise 7.1. 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 inter-arrival 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. Assume that the travel time between the two stations must be modeled. 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.

From your models, 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 20000 minutes with a warm up period of 5000 minutes.

Exercise 7.2. Reconsider part (b) of Exercise 7.1. Instead of immediately moving the part, the transport worker waits until a batch of 5 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 20000 minutes with a warm up period of 5000 minutes.

Exercise 7.3. Redo Exercise 4.25 using resource constrained transfer. Assume that there are 2 workers at the diagnostic station, 1 worker per testing station, and 3 workers at the repair station. Thus, there are a total of 8 workers in the system. Furthermore, assume that any of these 8 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

7. Modeling Systems with Entity Movement and Material Handling Constructs

assist the company in assessing the risks associated with the new contract under this resource constrained situation.

Exercise 7.4. Redo Exercise 4.26 assuming that there is a pool of 3 workers that 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.

Exercise 7.5. Redo Exercise 5.17 assuming that there is a pool of 3 workers that 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.

Exercise 7.6. In Section 7.2.1 the test and repair system was analyzed with 3 transporters. Re-analyzed this situation and recommend an appropriate number of transport workers.

Exercise 7.7. Reconsider part (b) of Exercise 7.1 Instead of modeling the problem with a resource, the problem should be modeled with a transporter. Assume that there is 1 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 20000 minutes with a warm up period of 5000 minutes.
 - b. Instead of immediately moving the part, the fork truck waits until a batch of 5 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.
-

Exercise 7.8. Reconsider Exercise 7.4. 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	A	B	C	D	Exit
A	—	50	60	80	90
B	55	—	25	55	75
C	60	20	—	70	30
D	80	60	65	—	35
Exit	100	80	35	40	—

There are 2 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 30000 minutes and discuss the potential bottlenecks in the system.

Exercise 7.9. Reconsider the Test and Repair example from Section 7.2.1 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.

Station	Diagnostics	Test 1	Test 2	Test 3	Repair	Home base
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.

Exercise 7.10. Reconsider Exercise 6.12 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:

Station	1	2	3	Exit
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 down stream 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 6.12, estimate the effect of the transporters on the throughput of the system.

Exercise 7.11. 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 inter-arrival 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.

Exercise 7.12. Reconsider Exercise 6.12 with conveyors to move the parts between the stations. Suppose that a single non-accumulating 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 6.12, estimate the effect of the conveyors on the throughput of the system.

Exercise 7.13. This problem considers the use of AGVs for the Test and Repair System of this chapter. The layout of the proposed system is given in Figure 7.52 with all measurements in meters.

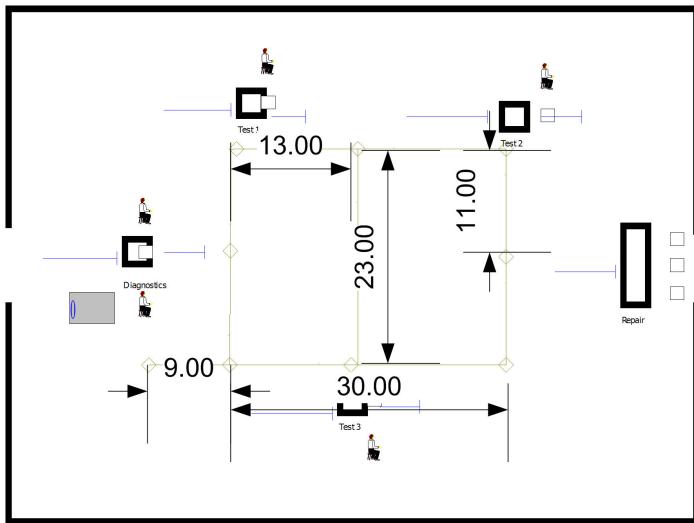


Figure 7.52.: Proposed system layout

There is only 1 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.

- 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.
- Consider the possibility of having a 2 AGV system. What changes to your design do you recommend? Be specific enough that you could simulate your design.

Exercise 7.14. (*This problem is based on an example on page 223 of (Pegden et al., 1995). Used with permission*) Reconsider Exercise 5.18 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.

7. Modeling Systems with Entity Movement and Material Handling Constructs

	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 two 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.

Exercise 7.15. (*This problem is based on an example on page 381 of (Pegden et al., 1995). Used with permission*). Reconsider Exercise 7.14 with the use of AGVs. There are now 3 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 in Figure 7.53:

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.

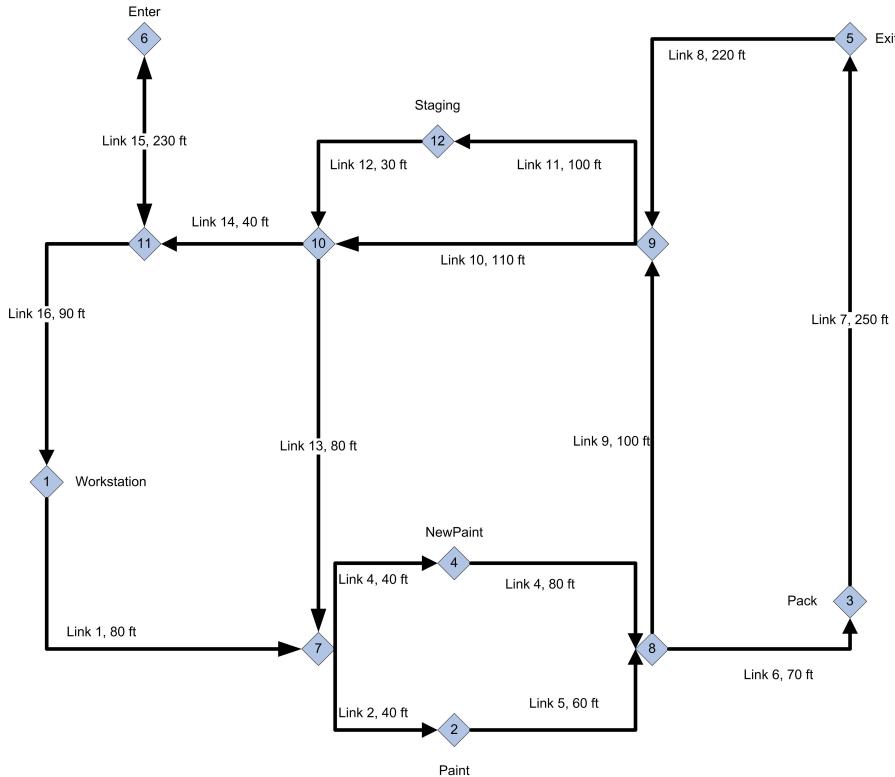


Figure 7.53.: AGV system layout

Exercise 7.16. 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	Travel distance to next station
Machine 1	2 per minute	uniform(3,5) seconds	40 meters
Machine 2	1 per minute	uniform(2,5) seconds	28 meters
Machine 3	3 per minute	uniform(4,6) seconds	55 meters

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.

7. Modeling Systems with Entity Movement and Material Handling Constructs

- 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%.
-

8. Applications of Simulation Modeling

LEARNING OBJECTIVES

- To be able to apply simulation to analyze multiple design alternatives in a statistically valid manner.
- 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.

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

1. Define the system and the problem
2. Establish performance metrics
3. Build conceptual model
4. Document modeling assumptions

2. Simulation Model Building

1. Model translation
2. Input data modeling
3. Verification and validation

3. Experimental Design and Analysis

1. Preliminary runs
2. Final experiments
3. Analysis of results

4. Evaluate and Iterate

5. Documentation

6. Implementation of results

8. Applications of Simulation Modeling

To some extent, our study of simulation has followed these general steps. Chapter 2 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 4 provided the primary modeling constructs to enable the translation of a conceptual model to a simulation model within Arena. In addition, Chapters 6 and 7 delved deeper into a variety of modeling situations to round out the tool set of modeling concepts and constructs that you can bring to bear on a problem.

Since simulation models involve randomness, Appendix B 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, Chapters 3 and 5 showed how to analyze the statistical aspects of simulation. However, there is one topic that we have yet to discuss that is useful when performing realistic simulation studies, comparing multiple alternatives or scenarios. Once you have a practical grasp of how to compare multiple system configurations in a valid manner, you will be all set to analyze realistic models. To prepare you for this topic and the rest of this chapter, we will revisit the LOTR Makers model of Chapter 4 in order to apply these new concepts within a familiar situation. Then, we will apply the techniques with a new modeling context on a larger, more realistic system. Let's get started with the topic of comparing multiple systems.

8.1. 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 will also illustrate how to perform a multiple comparison with the best analysis using the Process Analyzer.

8.1.1. 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 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 8.1.

Table 8.1.: Sensitivity Analysis Factors/Levels

Factor	Description	Base Level	% Change	Low Level	High Level
A	# sales calls per day	100	± 10 %	90	110
B	Std. Dev. of outer ring diameter	0.005	± 2 %	0.0049	0.0051
C	Std. Dev. of inner ring diameter	0.002	± 2 %	0.00196	0.00204

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 4 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 percent change as the level of the factor

8. Applications of Simulation Modeling

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 8.2 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 achieving this would be to define an EXPRESSION and use the expression within the model. For example, you can define an expression `eNumCalls = 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.

Table 8.2.: 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

The Process Analyzer is a separate program that is installed when the 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 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 8.1.

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.

8.1. Analyzing Multiple Systems

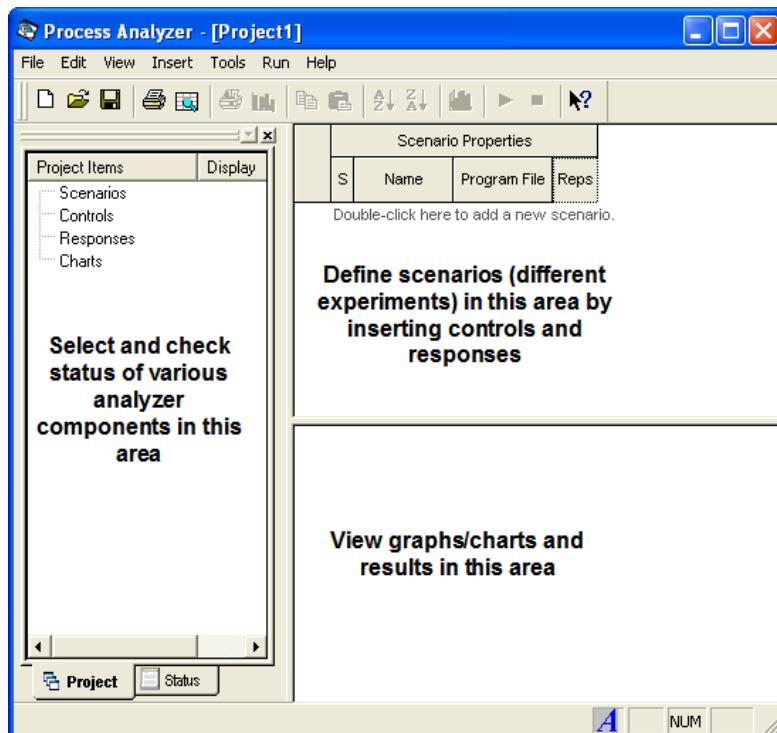


Figure 8.1.: The Process Analyzer

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 8.2.

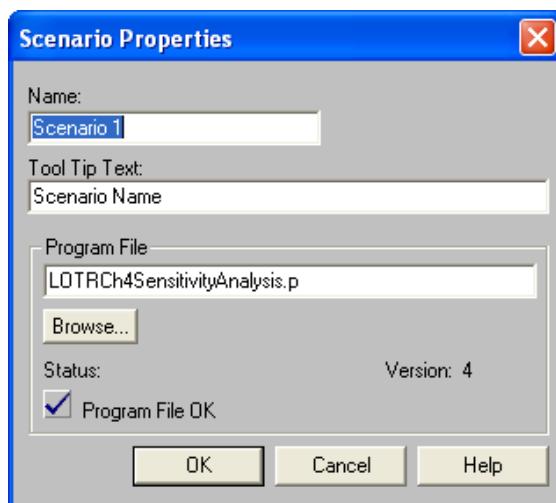


Figure 8.2.: Adding a new scenario

Then you can use the Insert menu to insert controls and response variables for the scenario. The

8. Applications of Simulation Modeling

insert control dialog is shown in Figure 8.3. Using this you should define a control for NREPS (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, vIDF is a real).

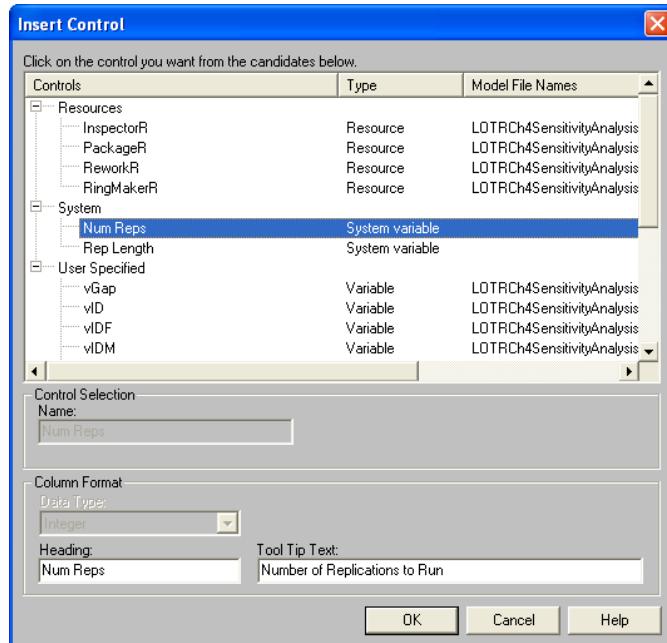


Figure 8.3.: Inserting controls

After specifying the first scenario, select the row, right-click and choose Duplicate Scenario(s) until you have defined 8 scenarios. Then, you should complete the scenario definitions as shown in Figure 8.4. 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).

S	Scenario Properties			Controls					Responses		
	Name	Program File	Reps	Num Reps	vStream	vNCF	vIDF	vODF	Rework.Queue.WaitingTime	ThroughPut	PairOfRings.TotalTime
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 8.4.: Experimental setup controls and responses

8.1. Analyzing Multiple Systems

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 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 in Figure 8.5. After the simulations have been completed, you can add more responses and the results will be shown.

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	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 8.5.: Results after running the scenarios

The purpose in performing a sensitivity analysis is two-fold: 1) to see if small changes in the factors result in significant changes in the responses and 2) 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 8.5 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.

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 ANOVA tests of differences in means are not really meaningful. In other words,

8. Applications of Simulation Modeling

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 8.5 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 8.6.

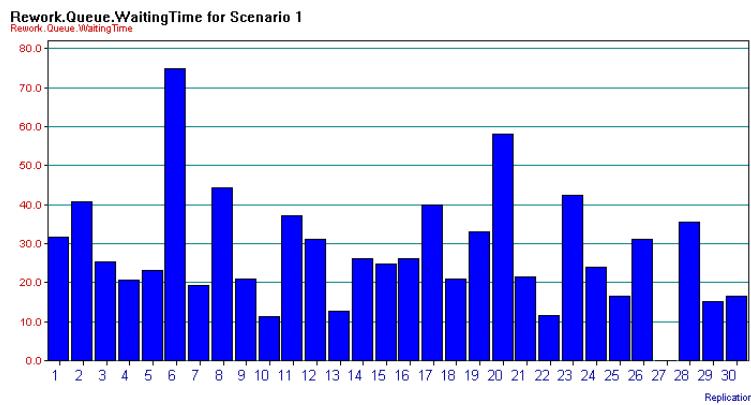
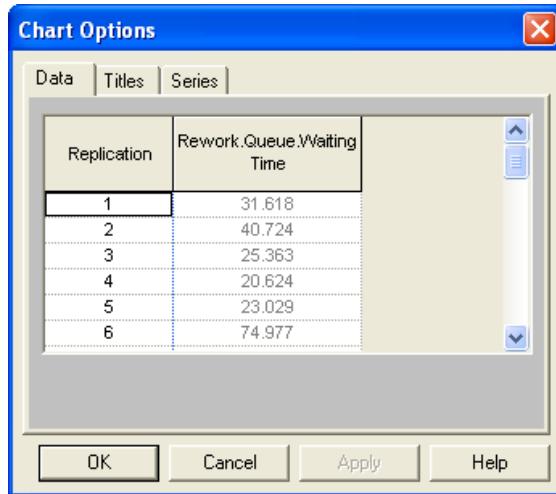
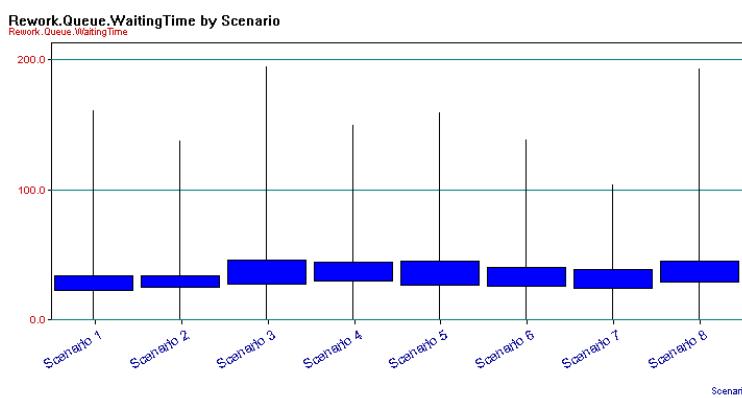


Figure 8.6.: Individual scenario chart for rework waiting time

If you right-click on the chart options pop-up menu, you can gain access to the data and properties of the chart (Figure 8.7). 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 whisker chart of the waiting times. Follow the wizard through the process until you have created a chart as shown in Figure 8.8.

There are varying definitions of what constitutes a box-whiskers plot. In the Process Analyzer, the box-whiskers plot shows whiskers (narrow thin lines) to the minimum and maximum of the

**Figure 8.7.:** Data within chart options**Figure 8.8.:** Box-Whiskers chart across scenarios

data and a box (dark solid rectangle) which encapsulates the inter-quartile range for the data (3^{rd} Quartile– 1^{st} 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 give 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 8.3 shows the data copied from the box-whiskers chart.

Table 8.3.: Data from Box-Whiskers Chart

Scenario	Min (A)	Max	Low	Hi	95% CI
Scenario 1	0	160.8	22.31	33.42	5.553
Scenario 2	0	137.2	24.93	33.69	4.382
Scenario 3	0	194.1	27.47	45.88	9.205

8. Applications of Simulation Modeling

Scenario	Min (A)	Max	Low	Hi	95% CI
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

This section illustrated how to setup 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 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.

8.1.2. Multiple Comparisons with the Best

Suppose that you are interested in analyzing k systems based on performance measures θ_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 θ_i must be estimated, which results in sampling error for each individual estimate of θ_i . The decision will be based upon all the estimates of θ_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:

$$P\{\cap_{i=1}^k E_i\} \geq 1 - \sum_{i=1}^k \alpha_i$$

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.

8.1. Analyzing Multiple Systems

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.

$$P\{\text{all } S_i \text{ true}\} = P\{\cap_{i=1}^k E_i\} \geq 1 - \sum_{i=1}^k \alpha_i = 1 - \alpha_E$$

where $\alpha_E = \sum_{i=1}^k \alpha_i$. The value α_E is called the overall error probability. This statement can be restated in terms of its complement event as:

$$P\{\text{one or more } S_i \text{ are false}\} \leq \alpha_E$$

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

$$\alpha_E = \sum_{i=1}^{10} \alpha_i = \sum_{i=1}^{10} (0.1) = 1.0$$

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 zero 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 zero and one. 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 pre-specify the α_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 α be to get $\alpha_E = 0.05$? Assuming that you have c confidence intervals, this yields:

$$\alpha_E = \sum_{i=1}^c \alpha_i = \sum_{i=1}^c \alpha = c\alpha$$

8. Applications of Simulation Modeling

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:

$$P\{\text{all } S_i \text{ true}\} = \prod_{i=1}^c (1 - \alpha_i)$$

If you are comparing k systems where one of the systems is the standard (e.g. base case, existing system, etc.), you can reduce the number of confidence intervals by analyzing the difference between the other systems and the standard. That is, suppose system one 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

$$c = \binom{k}{2} = \frac{k(k-1)}{2}$$

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 pre-specified 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.

While it is beyond the scope of this textbook to review multiple comparison with the best (MCB) procedures, it is useful to have a basic understanding of the form of the confidence interval constructed by these procedures. To see how MCB procedures avoid part of the issues with having a large number of confidence intervals, we can note that the confidence interval is based on the difference between the best and the best of the rest. Suppose we have k system configurations to compare and suppose that somehow you knew that the i^{th} system is the best. Now, consider a confidence interval system's i performance metric, θ_i , of the following form:

$$\theta_i - \max_{j \neq i} \theta_j$$

This difference is the difference between the best (θ_i) and the second best. This is because if θ_i is the best, when we find the maximum of those remaining, it will be next best (i.e. second best). Now, let us suppose that system i is *not* the best. Reconsider, $\theta_i - \max_{j \neq i} \theta_j$. Then, this difference will represent the difference between system i , which is not the best, and the best of the remaining systems. In this case, because i is not the best, the set of systems considered in $\max_{j \neq i} \theta_j$ will contain the best. Therefore, in either case, this difference:

$$\theta_i - \max_{j \neq i} \theta_j$$

tells us exactly what we want to know. This difference allows us to compare the best system with the rest of the best. MCB procedures build k confidence intervals:

$$\theta_i - \max_{j \neq i} \theta_j \text{ for } i = 1, 2, \dots, k$$

Therefore, only k confidence intervals need to be considered to determine the best rather than, $\binom{k}{2}$.

This form of confidence interval has also been combined with the concept of an indifference zone. Indifference zone procedures use a parameter δ that indicates that the decision maker is indifferent between the performance of two systems, if the difference is less than δ . This indicates that even if there is a difference, the difference is not practically significant to the decision maker within this context. These procedures attempt to ensure that the overall probability of correct selection of the best is $1 - \alpha$, whenever, $\theta_{i^*} - \max_{j \neq i^*} \theta_j > \delta$, where i^* is the best system and δ is the indifference parameter. This assumes that larger is better in the decision making context. There are a number of single stage and multiple stage procedures that have been developed to facilitate decision making in this context.

One additional item of note to consider when using these procedures. The results may not be able to distinguish between the best and others. For example, assuming that we desire the maximum (bigger is better), then let \hat{i} be the index of the system found to be the largest and let $\hat{\theta}_i$ be the estimated performance for system i , then, these procedures allow the following:

- If $\hat{\theta}_{\hat{i}} - \hat{\theta}_{\hat{i}} + \delta \leq 0$, then we declare that system i is not the best.
- If $\hat{\theta}_{\hat{i}} - \hat{\theta}_{\hat{i}} + \delta > 0$, then we can declare that system i is not statistically different from the best. In this case, system i , may in fact be the best.

The procedure used by Arena's Process Analyzer allows for the opportunity to provide an indifference parameter and the procedure will highlight via color which systems are best or not statistically different from the best. The statistical software, *R*, also has a package that facilitates multiple comparison procedures called *multcomp*¹. The interested reader can refer to the references for the package for further details.

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%.

¹<https://cran.r-project.org/web/packages/multcomp/index.html>

8. Applications of Simulation Modeling

The procedure built into the Process Analyzer can handle common random numbers. The previously described scenarios were set up and re-executed as shown in Figure 8.9. 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 folder called *SensitivityAnalysis* for this chapter.

S	Scenario Properties				Controls					Responses	
	Name	Program File	Reps	Num Reps	vStream	vNCF	vIDF	vODF	Rework.QueueWaitingTime	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	314.636
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 8.9.: Results for MCB analysis using common random numbers

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 as well 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 3rd wizard step you can adjust your titles for your chart. When you get to the 4th wizard step, you have the option of identifying the best scenario. Figure 8.10 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, & 4) as shown in Figure 8.11.

As indicated in Figure 8.11, four possible scenarios have been recommended as the best. This means that you can be 95% confident that any of these 4 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.

Now that we have a good understanding of how to effectively use the Process Analyzer, we will

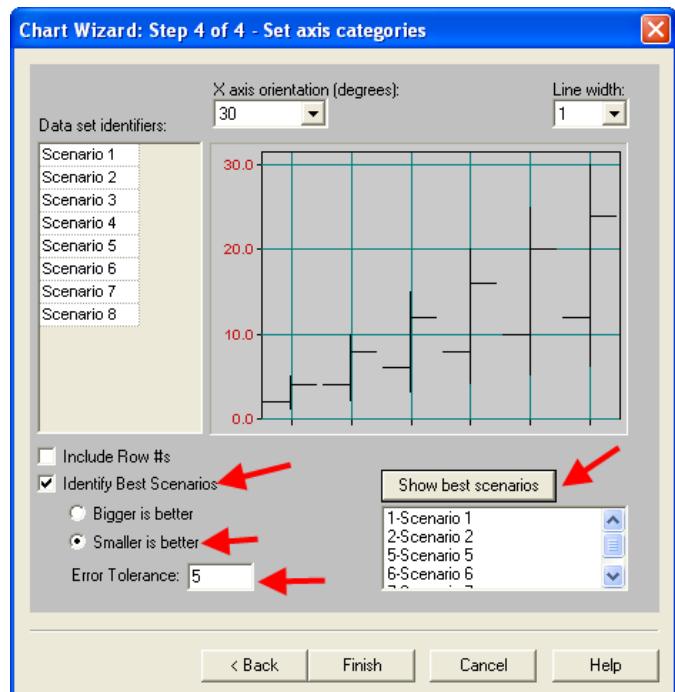


Figure 8.10.: Identifying the best scenario

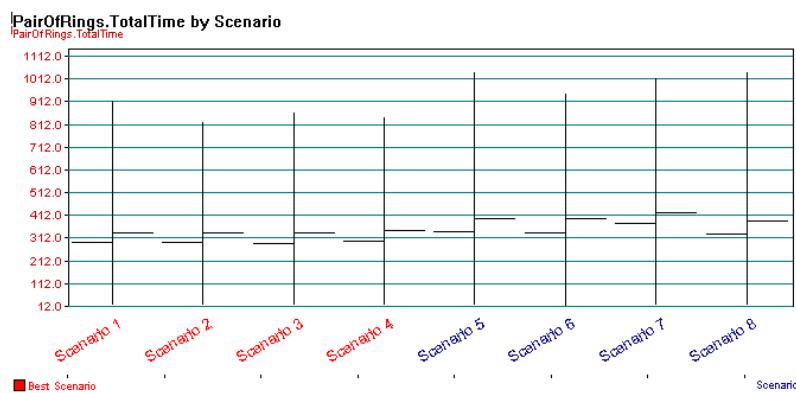


Figure 8.11.: Possible best scenarios

8. Applications of Simulation Modeling

study how to apply these concepts on an Arena Contest problem.

8.2. SM Testing Contest Problem Description

At this point you have learned a great deal concerning the fundamentals of simulation modeling and analysis. The purpose of this section 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) and the Institute of Industrial Engineers (IIE) have sponsored a student contest involving the use of to model a realistic situation. The contest problems have been released to the public and can be found on the main Arena website². This section will solve 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.

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

²<https://www.arenasimulation.com/>

8.2. SM Testing Contest Problem Description

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 8.12.

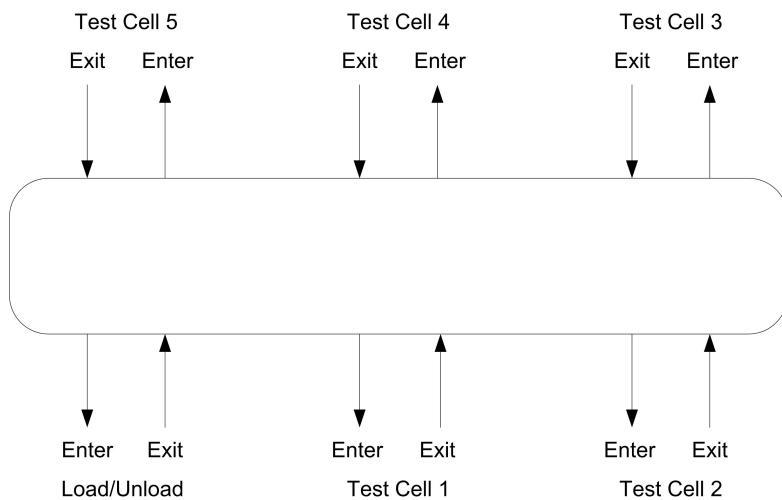


Figure 8.12.: Schematic of standard configuration

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. 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

8. Applications of Simulation Modeling

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's 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 an on 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's 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 5 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 provided in Figure 8.13.

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 ar-

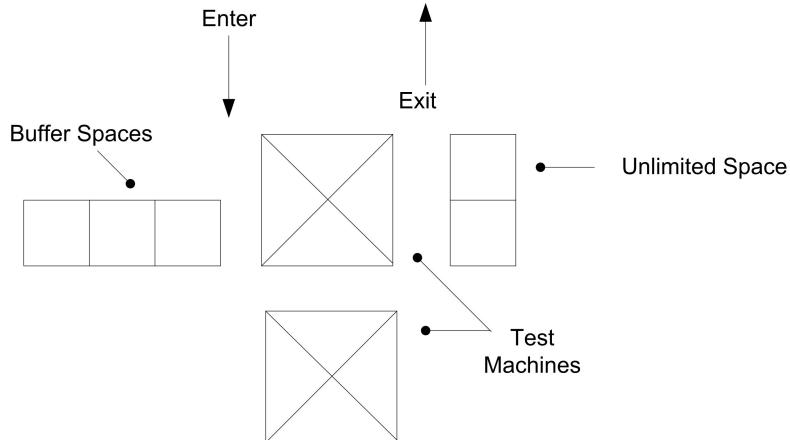


Figure 8.13.: Test cell schematic

rived 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 5 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 8.14.

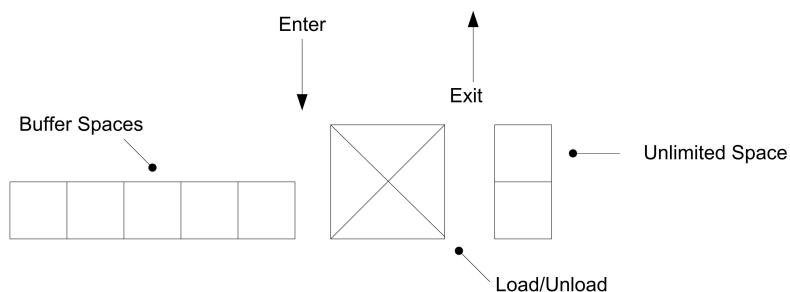


Figure 8.14.: Load and unload cell schematic

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.
- The sample holder is empty and a new sample is available. In this case, the new sample is

8. Applications of Simulation Modeling

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 8.4.

Table 8.4.: Tester cycle times in minutes

Tester	Time
1	0.77
2	0.85
3	1.03
4	1.24
5	1.7

Each test performed at Tester 3 requires 1.6 oz of reagent, and 38% of the tests at Tester 4 require 0.6 oz 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 8.5 for four of the testers. The units for mean time between failures (MTBF) are hours, and the units for mean time to repair (MTTR) are minutes.

Table 8.5.: Tester failure time (hours) and repair times (minutes)

Tester	MTBF	MTTR
1	14	11
3	9	7
4	15	14
5	16	13

The testers for Test Cell 2 rarely fail, but they do require cleaning after performing 300 tests. The

8.2. SM Testing Contest Problem Description

clean time follows a triangular distribution with parameters (5, 6, 10) (minutes).

The next pilot-testing laboratory will be open 24 hours a day, seven days a week. Our staff has projected demand data for the site, which are provided Table 8.6. Hour 1 represents the time between midnight and 1AM. Hour 24 represents the time between 11 PM and midnight. The rate is expressed in average arrivals per hour. The samples arrive without interruption throughout the day at these rates.

Table 8.6.: Sample arrival rates per hour

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

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 8.7.

Table 8.7.: Test sequences

#	Test Cells	(%)
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

Our contracts generally require that we provide test results within one 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

8. Applications of Simulation Modeling

given in Table 8.8 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 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.

Table 8.8.: Tester and sample holder costs

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

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 two months from now. Since there are several groups competing for this contract, we

8.3. Answering the Basic Modeling Questions

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 receiving your report on time and reviewing your proposed solution.

8.3. 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 than what is presented here, your effort and engagement in the problem will be important to how much you get out of this process. Let's 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 8.4.
- Time to failure and repair distributions for testers 1, 3-5. See Table 8.5.
- 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 8.6.
- 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.

8. Applications of Simulation Modeling

From this initial review of the information and the contest problem description, you should be able to develop an initial list of the 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 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, test result criteria, etc.

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

8.3. Answering the Basic Modeling Questions

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 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:

8. Applications of Simulation Modeling

- **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 (a.k.a. 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 flow chart). Although in many of the previous modeling examples, there was almost a direct mapping from the conceptual model to the 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.

8.3. Answering the Basic Modeling Questions

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 8.15 presents a simplified activity diagram for the samples. 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 8.16 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.

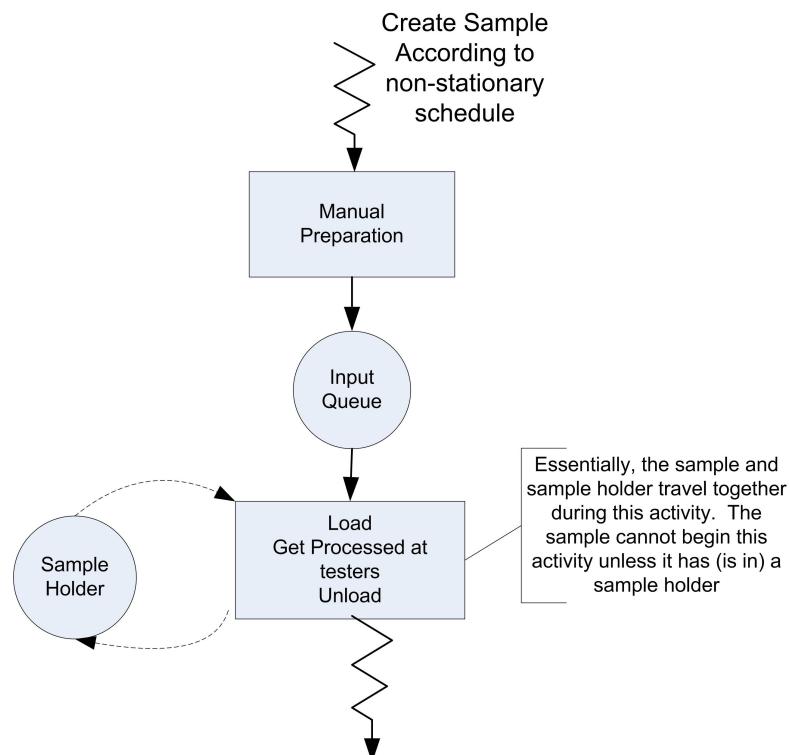


Figure 8.15.: Activity diagram for samples process

One thing to notice from this diagram is that the sequence does not have to be determined until

8. Applications of Simulation Modeling

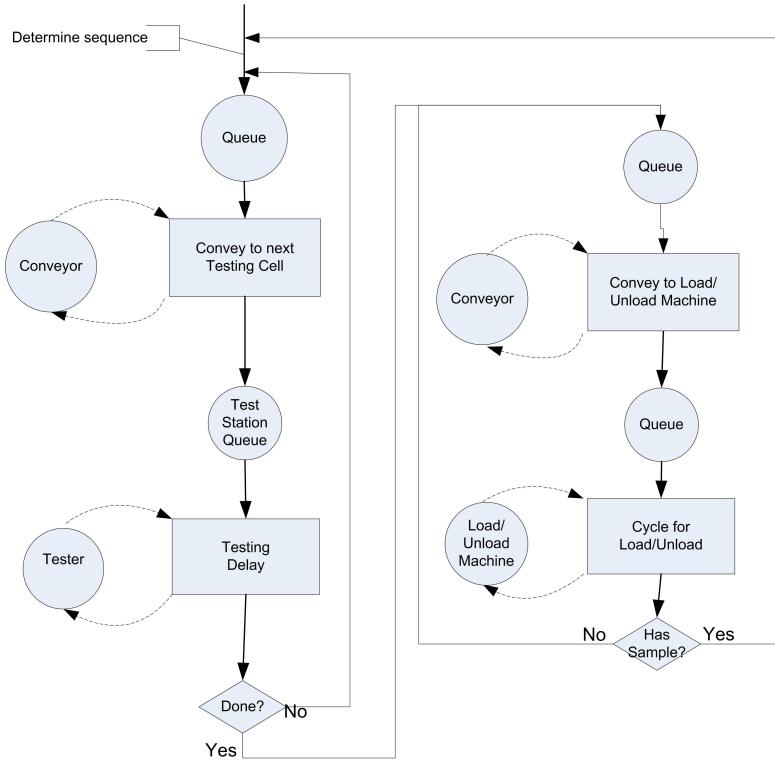


Figure 8.16.: Activity cycle diagram for sample holders

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.

8.4. 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 . 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.

8.4.1. 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 foot and 3 foot 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 foot 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.

Segment - Advanced Transfer

	Name	Beginning Station	Next Stations
1	Loop Conveyer.Segment	ExitLoadUnloadStation	12 rows

Next Stations

	Next Station	Length
1	TestCell1EnterStation	5
2	TestCell1ExitStation	3
3	TestCell2EnterStation	5
4	TestCell2ExitStation	3
5	TestCell3EnterStation	5
6	TestCell3ExitStation	3
7	TestCell4EnterStation	5
8	TestCell4ExitStation	3
9	TestCell5EnterStation	5
10	TestCell5ExitStation	3
11	EnterLoadUnloadStation	5
12	ExitLoadUnloadStation	3

Figure 8.17.: Modeling the conveyor segments

Figure 8.17 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

8. Applications of Simulation Modeling

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 8.18. 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.

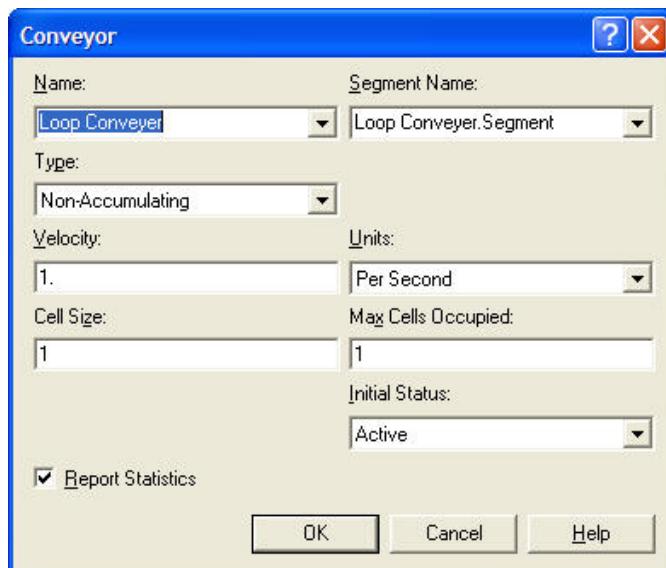


Figure 8.18.: Specifying the conveyor module

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 by sequence 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 8.19 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 8.20

The screenshot shows a software interface for defining sequences. On the left, a table lists 'Sequence' entries from 1 to 9, each with a 'Name' and 'Steps' count. On the right, a detailed view of 'Sequence 1' shows a table titled 'Steps' with columns: 'Station Name', 'Step Name', 'Next Step', and 'Assignments'. The first row is highlighted.

	Name	Steps
1	Sequence1	9 rows
2	Sequence2	7 rows
3	Sequence3	9 rows
4	Sequence4	7 rows
5	Sequence5	7 rows
6	Sequence6	9 rows
7	Sequence7	9 rows
8	Sequence8	7 rows
9	Sequence9	7 rows

	Station Name	Step Name	Next Step	Assignments
1	TestCell1EnterStation			0 rows
2	TestCell1ExitStation			0 rows
3	TestCell2EnterStation			0 rows
4	TestCell2ExitStation			0 rows
5	TestCell3EnterStation			0 rows
6	TestCell3ExitStation			0 rows
7	TestCell4EnterStation			0 rows
8	TestCell4ExitStation			0 rows
9	EnterLoadUnloadStation			0 rows

Figure 8.19.: Specifying the job steps for test sequence 1

shows the Advanced Set module used to define the set of sequences for the model. In addition, Figure 8.21 shows the implementation of the distribution across the sequences as an expression, called `eSeqIndex`, using the `DISC()` function within the EXPRESSION module.

The screenshot shows a software interface for defining sets. On the left, a table lists 'Set' entries from 1 to 5, each with a 'Name', 'Set Type', and 'Members' count. On the right, a detailed view of 'SequenceSet' shows a table titled 'Members' with a single column 'Other' containing the sequence names from 1 to 9.

	Name	Set Type	Members
1	TesterQueueSet	Queue	5 rows
2	EnterTestingStationSet	Other	5 rows
3	ExitTestingStationSet	Other	5 rows
4	ConveyorQueueSet	Queue	5 rows
5	SequenceSet	Other	9 rows

	Other
1	Sequence1
2	Sequence2
3	Sequence3
4	Sequence4
5	Sequence5
6	Sequence6
7	Sequence7
8	Sequence8
9	Sequence9

Figure 8.20.: Filling the set for holding the sequences

The screenshot shows a software interface for defining expressions. On the left, a table lists 'Expression' entries from 1 to 5, each with a 'Name' and 'Expression Values' count. On the right, a detailed view of 'eSeqIndex' shows an expression value of `DISC(0.09, 1, 0.22, 2, 0.37, 3, 0.49, 4, 0.56, 5, 0.67, 6, 0.81, 7, 0.87, 8, 1.0, 9)`.

	Name	Rows	Columns	Expression Values
1	eTestCycleTime	5		5 rows
2	eLoadUnloadTime			1 rows
3	eManualPrepTime			1 rows
4	ePriority			1 rows
5	eSeqIndex			1 rows

	Expression Values
	DISC(0.09, 1, 0.22, 2, 0.37, 3, 0.49, 4, 0.56, 5, 0.67, 6, 0.81, 7, 0.87, 8, 1.0, 9)

Figure 8.21.: Discrete distribution for assigning random sequences

8.4.2. 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

8. Applications of Simulation 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 8.16. This is essentially what happens to the sample when it is in the black box of Figure 8.15. Thus, to simplify the modeling, 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.

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 the enter load/unload area, where it exits the conveyor and is disposed. The following pseudo-code illustrates these concepts.

```
CREATE sample according to non-stationary pattern
BEGIN ASSIGN
    myEnterSystemTime = TNOW
    myPriorityType ~ Priority CDF
END ASSIGN
DELAY for manual preparation time
ROUTE to ExitLoadUnloadStation

STATION ExitLoadUnLoadStation
BEGIN 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
        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
    ENDIF
END DECIDE

STATION Generic Testing Cell Exit
ACCESS Loop Conveyor
CONVEY by Sequence

STATION EnterLoadUnloadStation
EXIT Loop Conveyor
RECORD System time
DISPOSE

```

Given the pseudo-code and the previous modeling, you should be able to develop an initial model for this simplified situation. For practice, you might try to implement the ideas that have been discussed before proceeding.

The model (with animation) representing this initial modeling is given in the file, *SMTTestingInitialModeling.doe*. The flow chart modules corresponding to the pseudo-code are shown in Figure 8.22.

The following approach was taken when developing the initial model for samples and test cells:

- *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

8. Applications of Simulation Modeling

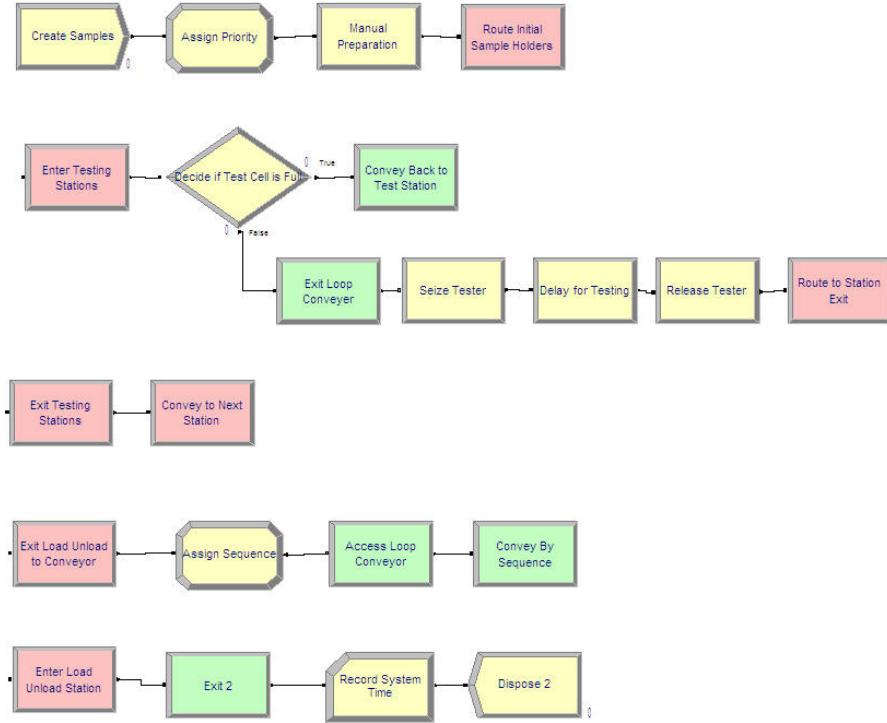


Figure 8.22.: Initial model for samples and test cells

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 9 members 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 9 sequences followed by the samples.
- *Schedules* An ARRIVAL schedule (see Figure 8.23) 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 8.24. 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.

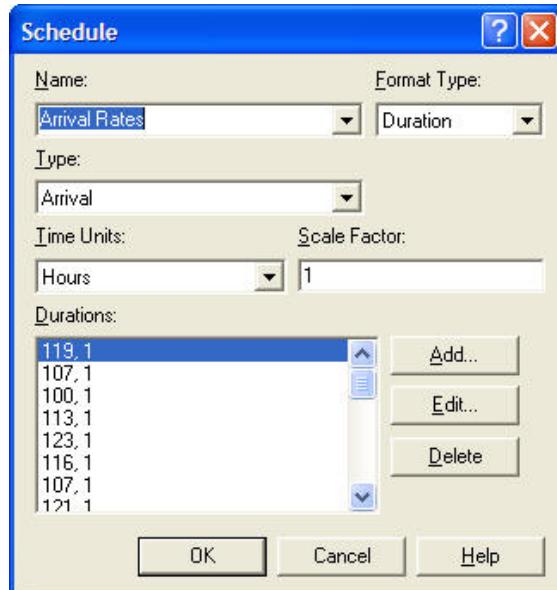


Figure 8.23.: ARRIVAL schedule for samples

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		EXPO(11)	Minutes	
2	Tester 3 Failure	Time	EXPO(9)	Hours		EXPO(7)	Minutes	
3	Tester 4 Failure	Time	EXPO(15)	Hours		EXPO(14)	Minutes	
4	Tester 5 Failure	Time	EXPO(16)	Hours		EXPO(13)	Minutes	
5	Tester 2 Failure	Count	1.0	Hours	300	TRIA(5 , 6 , 10)	Minutes	

Figure 8.24.: FAILURE modules for testers

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 8.9. The time to travel 33 feet at 1 foot/second is 0.55 minutes.

8. Applications of Simulation Modeling

Table 8.9.: Single sample system times.

Sequence	Steps	Distance	Feet			Time in Minutes	
			Travel	Test	Preparation	Total	
1	1-2-4-5	33	0.55	5.11	5	10.11	
2	3-4-5	36	0.60	3.97	5	9.57	
3	1-2-3-4	33	0.55	3.89	5	9.44	
4	4-3-2	132	2.20	3.12	5	10.32	
5	2-5-1	84	1.40	3.32	5	9.72	
6	4-5-2-3	81	1.35	4.82	5	11.17	
7	1-5-3-4	81	1.35	4.74	5	11.09	
8	5-3-1	132	2.20	3.50	5	10.70	
9	2-4-5	36	0.60	3.79	5	9.39	

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 8.9. 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 file, *SMTTestingInitialModelingTest1Entity.doe*.

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 8.10.

Using the total percentages given in Table 8.10, 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 8.6, 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's look at the peak arrival rate first. From the discussion in Appendix C, the offered load is

Table 8.10.: Percentage of arrivals visiting each test cell.

Cell	Proportion from Sequence									Total
	1	2	3	4	5	6	7	8	9	
1	0.09	0.00	0.15	0.00	0.07	0.00	0.14	0.06	0.00	0.51
2	0.09	0.00	0.15	0.12	0.07	0.11	0.00	0.00	0.13	0.67
3	0.00	0.13	0.15	0.12	0.00	0.07	0.11	0.13	0.00	0.71
4	0.09	0.13	0.15	0.12	0.00	0.00	0.11	0.14	0.13	0.87
5	0.09	0.13	0.00	0.00	0.07	0.11	0.14	0.06	0.13	0.73

Table 8.11.: Offered load calculation for peak hourly rate.

Test	Testing Time		Rate per hour		Offered	
	Cell	Minutes	Hours	Arrival	Service	
1	1	0.77	0.01283	102	77.92	1.31
2	2	0.85	0.01417	134	70.59	1.90
3	3	1.03	0.01717	142	58.25	2.44
4	4	1.24	0.02067	174	48.39	3.60
5	5	1.70	0.02833	146	35.29	4.14

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 8.11 calculates the offered load for each of the test cells under the peak arrival rate.

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 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 8.25 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 8.12 indicates the offered load under the minimum arrival rate.

8. Applications of Simulation Modeling

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 8.25.: Results from offered load experiment

Table 8.12.: Offered load calculation for minimum hourly rate.

Test Cell	Testing Time		Rate per hour		Offered Load
	Minutes	Hours	Arrival	Service	
1	0.77	0.01283	51	77.92	0.65
2	0.85	0.01417	67	70.59	0.95
3	1.03	0.01717	71	58.25	1.22
4	1.24	0.02067	87	48.39	1.80
5	1.70	0.02833	73	35.29	2.07

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 8.13. The requirements were determined by rounding up the computed offered load for each test cell. This provides a 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.

Table 8.13.: Preliminary test cell resource requirements.

Test Cell	Resource Requirements	
	Low	High
1	1	2
2	1	2
3	2	3
4	2	4
5	3	5

8.4.3. 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 fact that sample holders wait in front of it and that it operates on sample holders indicates 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 four actions are:

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, an 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.

8. Applications of Simulation Modeling

- 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 7. 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 the following pseudo-code.

```
CREATE sample according to non-stationary pattern
BEGIN 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
        CONVEY back to EnterLoadUnloadStation
    ELSE
        IF sample holder is empty and new sample queue is empty
            CONVEY back to EnterLoadUnloadStation
        ELSE
            EXIT Loop Conveyor
            SEIZE load/unload machine
            DROPOFF sample if loaded
            DELAY for load/unload cycle time
            IF new sample queue is not empty
                PICKUP new sample
```

```

ENDIF
RELEASE load/unload machine
ROUTE to ExitLoadUnLoadStation
ENDIF
ENDIF
END DECIDE

STATION ExitLoadUnLoadStation
DECIDE
IF the sample holder has a sample
BEGINN ASSIGN
mySeqIndex ~ Sequence CDF
Entity.Sequence = MEMBER(SequenceSet, mySeqIndex)
Entity.JobStep = 0
END ASSIGN
ENDIF
END DECIDE
ACCESS Loop Conveyor
DECIDE
IF the sample holder has a sample
CONVEY by Sequence
ELSE
CONVEY to EnterLoadUnloadStation
ENDIF
END DECIDE

```

As seen in the pseudo-code, 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 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 conveyor empty until samples start to arrive. At which time, the logic will cause them to exit the conveyor to pick up samples.

8. Applications of Simulation Modeling

Figure 8.26 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 sub-models were used to model the test cells and the load/unload area. The logic for the test cell sub-model is exactly the same as described during the initial model development.

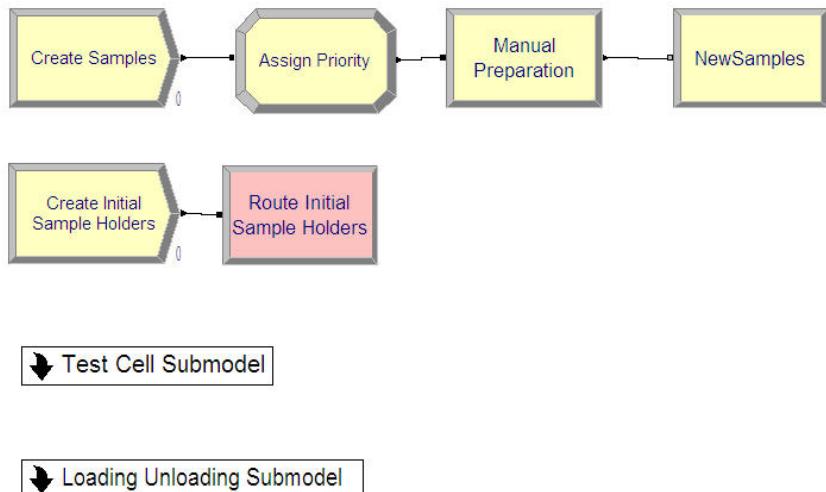


Figure 8.26.: Overview of entire model

The file, *SMTTesting.doe*, contains the completed model. The logic described in the previous pseudo-code is implemented in the sub-model, 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.

8.4.4. 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 sub-model for the load/unload area has an additional sub-model that implements the collection of these statistics using RECORD modules. Figure 8.27 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.

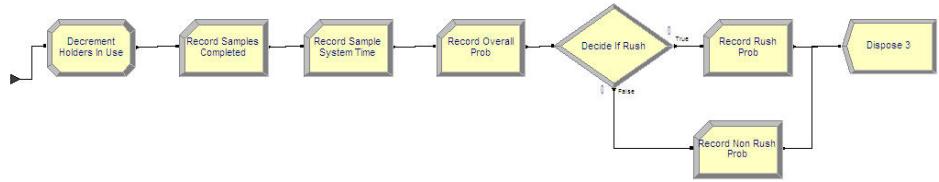


Figure 8.27.: System time statistical collection

The cost of each configuration can also be tabulated using output statistics (Figure 8.28). 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.

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 8.28.: STATISTICS module for collecting cost expressions

For example, Table 8.14 shows the total monthly cost calculation for the high resource case assuming the use of 16 sample holders.

Table 8.14.: 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

8. Applications of Simulation Modeling

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.

8.5. 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 3. 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's 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 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 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

8. Applications of Simulation Modeling

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 8.14 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 ten sample paths were averaged to produce a Welch plot as shown in Figure 8.29. 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.

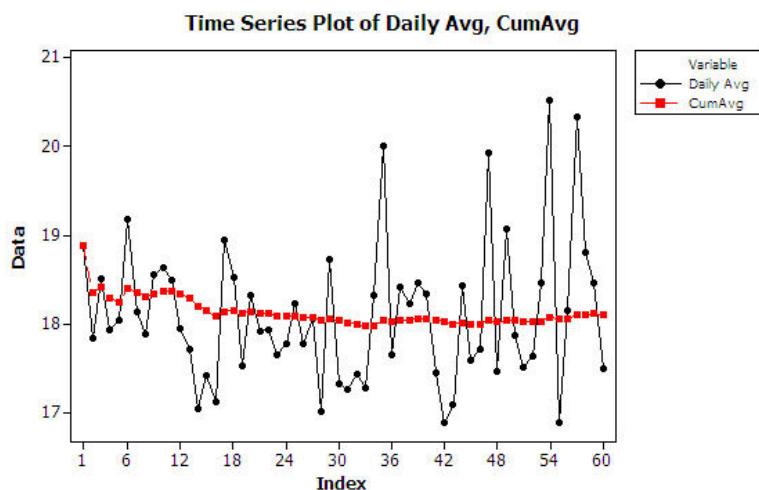


Figure 8.29.: Welch plot for high resource configuration

If we run the low resource case of Table 8.13 under the same conditions, an execution error will occur that indicates that the maximum number of entities for the professional version of 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.

The run setup parameter settings for the initialization bias investigation do not necessarily ap-

8.6. Preliminary Experimental Analysis

ply to further experiments. In particular, with the 3rd 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 1 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.

Tally		
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
Counter		
Count	Average	Half Width
SamplesCompleted	33688.00	150.40

Figure 8.30.: User defined results for pilot run

Using the high resource case with 48 sample holders a pilot run was made of 5 replications of 11 days with 1 day warm up. The user-defined results of this run are shown in Figure 8.30. 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 5 replications. Thus, 5 replications will be used in all the experiments that follow. Because other configurations may have more variability and thus require more than 5 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.

8.6. 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 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

8. Applications of Simulation Modeling

Table 8.15.: Initial results for cell and holder scenarios.

Resource Settings						Probability		System Time
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	#Holders	Non-Rush	Rush	(minutes)
2	2	3	4	5	48	0.998	0.960	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	0.990	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	0.990	39.989
2	2	3	4	5	16	0.494	0.989	57.565
2	2	3	4	5	12	0.000	0.988	1309.683
1	1	2	2	3	16	0.000	0.985	2525.434
1	1	2	2	3	48	0.000	0.749	2234.676

run). The execution of a woefully under capacitated system may cause an execution error that may require the experiments to have to be re-done. The analysis files associated with this and subsequent sections are available in the chapter files.

Based on Table 8.13, 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 8.15. 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.

8.7. 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 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 8.15. Then, after an OptQuest analysis has 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.

8.7.1. 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 (Montgomery and Runger, 2006). For a discussion of experimental design within a simulation context, please see (Law, 2007) or (Kleijnen, 1998). There are 6 factors (# units for each of the 5 testing cells and the number of sample holders). To initiate the analysis, a factorial experiment can be used. As

Table 8.16.: 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

shown in Table 8.16, 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 8.17. 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 cell's 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 8.18. 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 some what 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 8.15 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 8.19. This set of experiments is a $2^4 = 16$ factorial experiment. The results are shown in Table 8.20. 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

8.7. Final Experimental Analysis and Results

Table 8.17.: Half-fraction with number of 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	0.000	0.963	2410.517	100340
2	1	2	2	3	20	0.000	0.966	2333.135	110340
1	2	2	2	3	20	0.000	0.980	1913.694	112740
2	2	2	2	3	20	0.000	0.973	1904.499	122740
1	1	3	2	3	20	0.000	0.966	2373.574	108840
2	1	3	2	3	20	0.000	0.962	2407.404	118840
1	2	3	2	3	20	0.000	0.973	1902.445	121240
2	2	3	2	3	20	0.000	0.978	1865.821	131240
1	1	2	4	3	20	0.000	0.951	2332.412	119940
2	1	2	4	3	20	0.000	0.949	2365.047	129940
1	2	2	4	3	20	0.003	0.985	496.292	132340
2	2	2	4	3	20	0.025	0.986	284.205	142340
1	1	3	4	3	20	0.000	0.956	2347.050	128440
2	1	3	4	3	20	0.000	0.956	2316.931	138440
1	2	3	4	3	20	0.019	0.984	364.081	140840
2	2	3	4	3	20	0.122	0.987	157.798	150840
1	1	2	2	5	20	0.000	0.968	2394.530	122740
2	1	2	2	5	20	0.000	0.965	2360.751	132740
1	2	2	2	5	20	0.000	0.980	1873.405	135140
2	2	2	2	5	20	0.000	0.982	1865.020	145140
1	1	3	2	5	20	0.000	0.963	2391.649	131240
2	1	3	2	5	20	0.000	0.965	2361.708	141240
1	2	3	2	5	20	0.000	0.974	1926.889	143640
2	2	3	2	5	20	0.000	0.980	1841.387	153640
1	1	2	4	5	20	0.000	0.951	2387.810	142340
2	1	2	4	5	20	0.000	0.955	2352.933	152340
1	2	2	4	5	20	0.202	0.986	121.199	154740
2	2	2	4	5	20	0.683	0.991	43.297	164740
1	1	3	4	5	20	0.000	0.954	2291.880	150840
2	1	3	4	5	20	0.000	0.947	2332.031	160840
1	2	3	4	5	20	0.439	0.985	69.218	163240
2	2	3	4	5	20	0.961	0.991	21.431	173240

8. Applications of Simulation Modeling

Table 8.18.: Half-fraction with number of 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	0.000	0.776	2319.323	106532
2	1	2	2	3	36	0.000	0.764	2260.829	116532
1	2	2	2	3	36	0.000	0.729	1816.568	118932
2	2	2	2	3	36	0.000	0.714	1779.530	128932
1	1	3	2	3	36	0.000	0.768	2238.654	115032
2	1	3	2	3	36	0.000	0.775	2243.560	125032
1	2	3	2	3	36	0.000	0.730	1762.579	127432
2	2	3	2	3	36	0.000	0.717	1763.306	137432
1	1	2	4	3	36	0.000	0.796	2244.243	126132
2	1	2	4	3	36	0.000	0.790	2268.549	136132
1	2	2	4	3	36	0.202	0.900	111.232	138532
2	2	2	4	3	36	0.385	0.915	72.860	148532
1	1	3	4	3	36	0.000	0.806	2255.190	134632
2	1	3	4	3	36	0.000	0.795	2251.548	144632
1	2	3	4	3	36	0.360	0.899	76.398	147032
2	2	3	4	3	36	0.405	0.911	68.676	157032
1	1	2	2	5	36	0.000	0.771	2304.784	128932
2	1	2	2	5	36	0.000	0.771	2250.198	138932
1	2	2	2	5	36	0.000	0.738	1753.819	141332
2	2	2	2	5	36	0.000	0.717	1751.676	151332
1	1	3	2	5	36	0.000	0.780	2251.174	137432
2	1	3	2	5	36	0.000	0.771	2252.106	147432
1	2	3	2	5	36	0.000	0.726	1781.024	149832
2	2	3	2	5	36	0.000	0.716	1794.930	159832
1	1	2	4	5	36	0.000	0.787	2243.338	148532
2	1	2	4	5	36	0.000	0.794	2243.558	158532
1	2	2	4	5	36	0.541	0.897	54.742	160932
2	2	2	4	5	36	0.898	0.932	28.974	170932
1	1	3	4	5	36	0.000	0.791	2263.484	157032
2	1	3	4	5	36	0.000	0.799	2278.707	167032
1	2	3	4	5	36	0.604	0.884	49.457	169432
2	2	3	4	5	36	0.994	0.983	16.588	179432

8.7. Final Experimental Analysis and Results

Table 8.19.: 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 8.20.: Results for second set of experiments.

Cell Resource Settings					#Holders	Probability		System Time	Total Cost
1	2	3	4	5		Non-Rush	Rush		
2	2	3	4	5	24	0.988	0.990	16.355	174788
2	2	3	4	4	24	0.970	0.988	19.410	163588
2	2	3	4	5	20	0.961	0.991	21.431	173240
2	2	3	4	4	20	0.945	0.989	25.270	162040
2	2	3	3	4	24	0.877	0.987	30.345	153788
2	2	3	3	5	24	0.871	0.988	31.244	164988
2	2	2	4	5	24	0.812	0.984	34.065	166288
2	2	2	4	4	24	0.782	0.986	35.289	155088
2	2	3	3	5	20	0.734	0.991	39.328	163440
2	2	3	3	4	20	0.731	0.992	40.370	152240
2	2	2	3	4	24	0.687	0.987	42.891	145288
2	2	2	4	5	20	0.683	0.991	43.297	164740
2	2	2	3	5	24	0.656	0.980	46.049	156488
2	2	2	4	4	20	0.627	0.989	45.708	153540
2	2	2	3	5	20	0.561	0.987	52.710	154940
2	2	2	3	4	20	0.492	0.985	59.671	143740

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 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, ten 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 8.21. In addition, the multiple comparison procedure was applied

8. Applications of Simulation Modeling

Table 8.21.: Results for final set of experiments.

Scenario	Cell Resource Settings					#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	162040
2	2	2	3	4	4	21	0.942	0.987	23.273	162427
3	2	2	3	4	4	22	0.976	0.989	20.125	162814
4	2	2	3	4	4	23	0.974	0.988	19.983	163201
5	2	2	3	4	4	24	0.979	0.989	18.512	163588
6	2	2	3	4	5	20	0.958	0.988	21.792	173240
7	2	2	3	4	5	21	0.967	0.987	19.362	173627
8	2	2	3	4	5	22	0.984	0.988	18.022	174014
9	2	2	3	4	5	23	0.986	0.988	16.971	174401
10	2	2	3	4	5	24	0.980	0.987	16.996	174788

to the results based on picking the solution with the best (highest) non-rush probability. The results of that comparison are shown in Figure 8.31. The multiple comparison procedure indicates with 95% confidence that scenarios 3, 4, 5, 7, 8, 9, 10 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 applying OptQuest to a problem, let us examine its application to this situation.

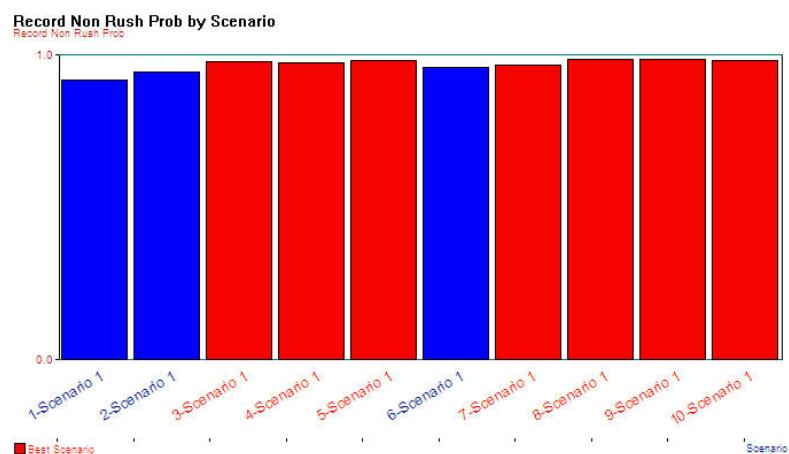


Figure 8.31.: Multiple comparison results for non-rush probability

8.7.2. Using OptQuest on the Problem

OptQuest is an add-on program for Arena which provides heuristic based simulation optimization capabilities. You can find *OptQuest* on *Tools > OptQuest*. *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, 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.

As shown in Figure 8.32, *OptQuest* for can be used to define controls and responses and to setup an optimization model to minimize total cost subject to the 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 8.33, 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.

8.7.3. 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 sub-model. In addition, a

8. Applications of Simulation Modeling

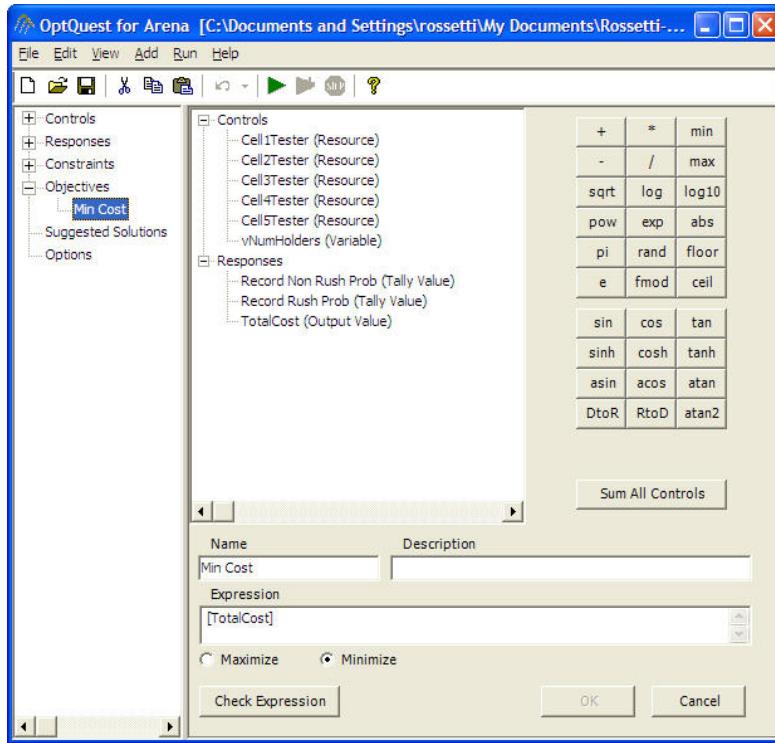


Figure 8.32.: Defining the objective function in OptQuest

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. TTable 8.22 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.

8.7. Final Experimental Analysis and Results

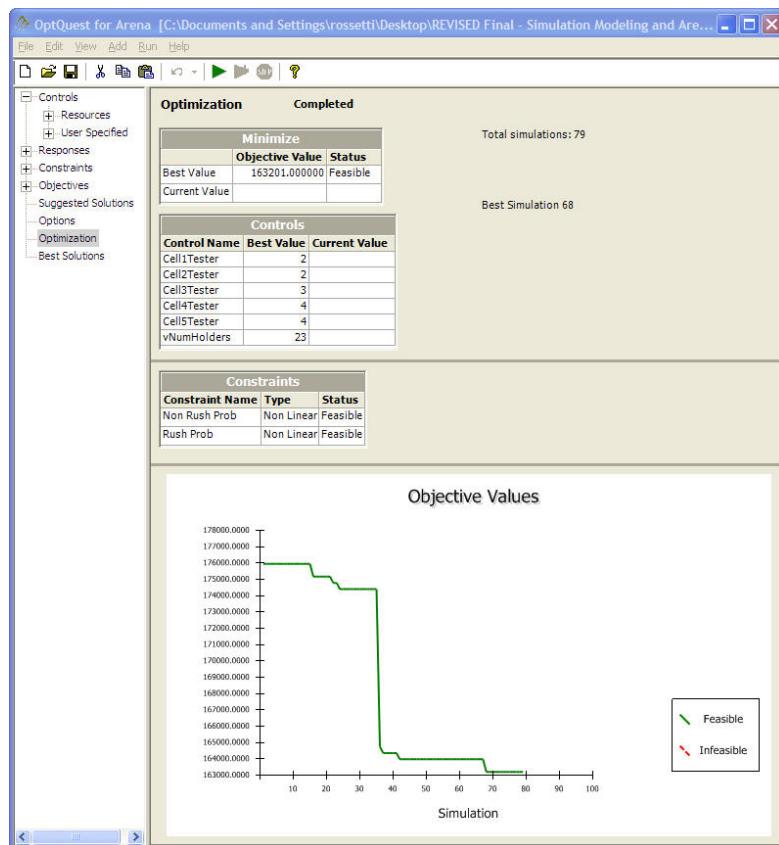


Figure 8.33.: Results from the OptQuest run

Table 8.22.: Results for analyzing the new logic.

Scenario	Cell Resource Settings					#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	18.170

8.7.4. Sensitivity Analysis

This realistically sized case study clearly demonstrates the application of simulation to designing 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, breakdowns, etc.) that were in the problem. Based on the analysis performed for the case study, SMTTesting should 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 that 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 re-test 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 previously in this chapter the mechanics of performing sensitivity analysis will not be discussed here. A number of exercises will ask you to explore these issues.

8.8. 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 model is especially useful in this regard. For the SMTTesting model, an animation is available within the file. The animation (Figure 8.34) 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.

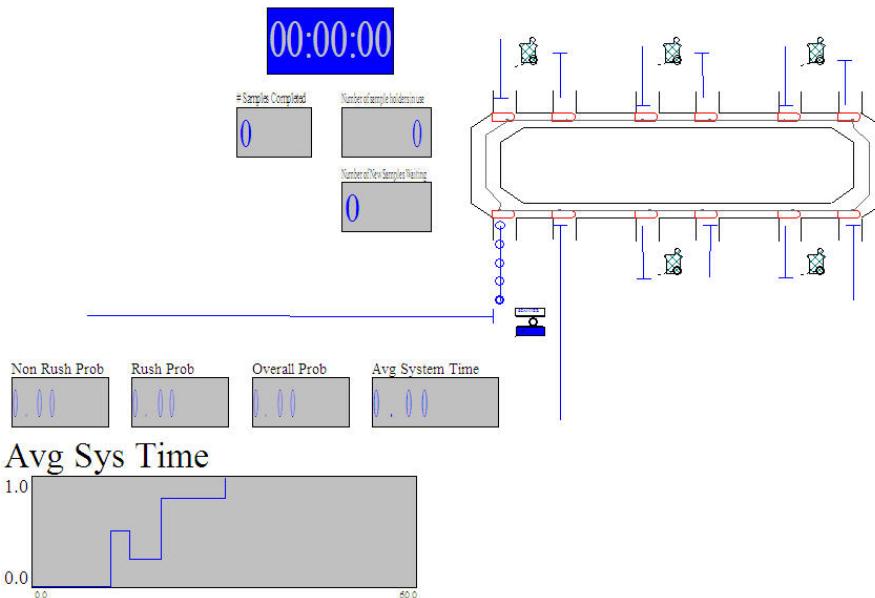


Figure 8.34.: Overview of the simulation animation

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 provided here.

1. Executive Summary

1. Provide a brief summary of the problem, method, results, and recommendation. It should be oriented towards the decision maker.
2. 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 after reading the summary.

2. Introduction

1. 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

1. 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

8. Applications of Simulation Modeling

description given in this chapter is an excellent example of describing the system and the problem.

4. Solution and Modeling Approach

1. 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.
2. 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.
3. State relevant and important assumptions. Discuss their potential effect on your answers and any procedures you took to test your assumptions.
4. 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, 1998).

5. Experimentation and Analysis

1. 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.
2. 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, sensitivity analysis and their effect on the model's results.

6. Conclusions and Recommendations

1. 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

1. Supporting materials. Detailed computer model documentation.

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 non-simulation oriented users will not be interacting with the model, it is very useful to have a users 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 re-discovered and used for additional analysis. A users guide becomes extremely useful in these situations.

8.9. 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, 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) illustrate 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)

8. Applications of Simulation Modeling

- 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 one.
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, normality, etc.) 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, sample half-width for a specified confidence level, etc.)
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 such techniques as debugging, event tracing, animation, etc.

8.9. Some Final Thoughts

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.

8.10. Exercises

Exercise 8.1. Perform a sensitivity analysis on the recommendation for the SMTTesting contest problem by varying the arrival rate for each hour by plus or minus 10%. In addition, examine the effect of decreasing or increasing the time between failures for test cells 1, 3, 4, 5 by plus or minus 10% as well as the mean time for repair.

Exercise 8.2. What solution should be recommended if the probability of meeting the contract requirements is specified as 95% for non-rush and rush samples?

Exercise 8.3. Revise the model to handle the independent failure of individual testers at each test cell. Does the more detailed modeling change the recommended solution?

Exercise 8.4. Revise the model to handle the situation of re-testing at each test cell. Assume that a re-test occurs with a 3% chance and then test the sensitivity of this assumption.

Exercise 8.5. How does changing the buffer capacities for the load/unload area and the test cells affect the recommended solution?

Exercise 8.6. 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.

Exercise 8.7. 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.

Exercise 8.8. This problem analyzes the cost associated with the LOTR, Inc. configurations described in Example 4.2 of Section 4.5. In particular, the master ring maker cost \$30 per hour whether she is busy or idle. Her time is considered value added. The rework craftsman earns \$22 per hour; however, his time is considered non-value 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 non-value 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 min 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, 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 Section 4.5 and report the entity and resource costs for the configurations. Which configuration would you recommend?

Exercise 8.9. Reconsider Exercise 5.10 of Chapter 5. 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 & 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.

8. Applications of Simulation Modeling

Base your analysis on 30 replications of length 1000 hours and a warm up period of 200 hours.

Exercise 8.10. Reconsider Exercise 4.24 of Chapter 4. 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 non-frequent flyers. Assign 3 of the current check-in agents to serve non-frequent 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 non-frequent flyers. One TSA agent is dedicated to non-frequent 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 non-frequent flyers. There are 3 check-in agents to serve non-frequent 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. Non-frequent flyer agents should be selected before the frequent flyer agent when serving the non-frequent 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 non-frequent 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 non-frequent flyers. One TSA agent is dedicated to non-frequent flyers. The other TSA agent is shared between the two. The TSA agent that 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 non-frequent flyers 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	Non-Frequent Flyer Total time in the system		
B	Frequent Flyer Total time in the system		
B	Non-Frequent Flyer Total time in the system		
C	Frequent Flyer Total time in the system		
C	Non-Frequent Flyer Total time in the system		

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.

Recommend with 95% confidence the best alternative A, B, or C for non-frequent flyers in terms of total time in the system. Use an indifference parameter of 0 minutes.

Exercise 8.11. In preparation for the vaccination of individuals, a major drug store chain is planning for the setup and operation of temporary vaccination clinics. In order to determine an efficient vaccination process, the company has decided to use discrete-event simulation modeling to evaluate various design alternatives and develop recommended base designs for a national roll out. Your project team has been asked to develop a simulation model that can be used by the company during the evaluation process and to recommend base configurations that meet some minimum performance requirements in a cost-effective manner. Alternative configurations may also be recommended for different operating conditions.

In general, an overview of the vaccination process is as follows:

- Patients (persons requiring a vaccination) arrive to the clinic. Additional details about the arrival process will be presented shortly.
- Upon arrival, the patient's pre-visit paperwork is checked to ensure that they have proper identification. If the arriving patient does not have the following: photo identification, insurance card, and scheduled visit confirmation, the patient will be asked to exit. After this check, the patient walks to the pre-vaccination area.
- After arriving to the pre-vaccination area, the patient waits for a temperature check, ensures that their paperwork is in order, and receives their vaccination lot control number. A staff member at the pre-vaccination area assists the patient with these tasks. Patients with temperatures above a threshold are asked to get a COVID test performed and to reschedule the vaccination, if necessary, on a different day. Patients might have arrived without a paper copy of their insurance or driver's license. If so, a photocopier is used to record a copy before proceeding. The staff member assisting the patient with their paperwork does not start on another patient until all tasks at the station are completed.
- After getting their pre-vaccination paperwork in order, the patient moves to one of the available seats within the pre-vaccination seating area, where they must wait in an area that has been set up for social distancing for an available vaccination nurse.
- When a vaccination nurse is available, the patient moves to a seat with a vaccination nurse. The vaccination includes the following time elements:
 - The nurse receives the patient paperwork, files the consent form, and records the pharmacy data and name on the vaccine record card.
 - The patient prepares for the injection by baring a preferred upper arm.
 - After changing to fresh gloves, the nurse swabs the upper arm of the patient.

8. Applications of Simulation Modeling

- The nurse injects the vaccination.
 - The nurse provides basic information about the vaccination and what to do next.
 - After the attending the patient, the nurse ensures that the area is clean for the next patient.
- After vaccination the patient collects their belongings and walks to the post-vaccination area.
 - At the post-vaccination area, a clerk checks if the patient wrote their name and date of birth on their vaccination card, informs the patient to use the scheduling link to schedule an appointment to return for the 2nd dose of the vaccine, and informs the patient about the 15-minute post-vaccine waiting period.
 - The patient must then sit in a waiting area for a required 15-minute time interval to ensure that there are no issues with an allergic reaction. The patients in the post-vaccination area must be monitored by at least one nurse for every 15 waiting patients.
 - If there are no issues during the post-vaccination waiting period, the patient can leave the clinic. Patients with an adverse reaction within the clinic are helped to a separate area and tended by emergency medical technicians.

While the vaccination process is relatively straightforward, there are a number of constraints, uncertainties, and issues that need to be addressed within the process to ensure that the process operates safely and effectively for all persons at the clinic.

The first assumption is that the process must be designed to meet a particular daily throughput. For example, the population, demographics, and stipulations of the contract may require that at least 600 people can be vaccinated per day over the period of time that the vaccination clinic operates. Generally, because of setup and cleaning operations, the clinics will be able to operate for at least 6 hours per day to justify operating and to meet their throughput requirement.

The second assumption is that the chain will roll out an on-line scheduling application that patients will be required to use to schedule their visit to the clinic. The scheduling application is used to control the number of people that will be processed throughout the day. The scheduling application permits the clinics operating hours (e. g. 6 hours) to be divided into time periods that can hold a fixed number of patients. For example, assuming that 600 people need to be scheduled in a 6-hour operating day, the 6 hours (360 minutes) can be divided into 36, 10-minute blocks of time for which about 16 or 17 patients can be scheduled. While the length, number of scheduling blocks, throughput requirement, and operating hours are input design parameters, the chain is explicitly interested in testing the following configuration:

- 600 patients per day
- 6 hours of operating time
- 36, 10-minute blocks

The design of the clinic should be based on basic service configurations involving the tables, chairs, and social distancing requirements. Two basic service table configurations are illustrated in Figures 8.35 and 8.36.

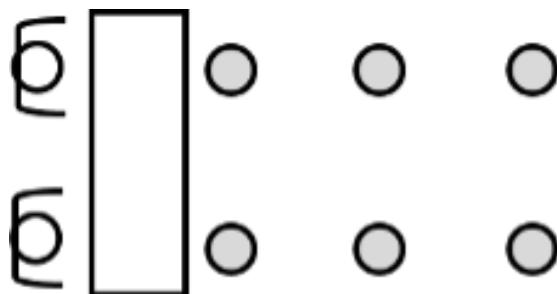


Figure 8.35.: Basic eight-foot table service station with 2 lines

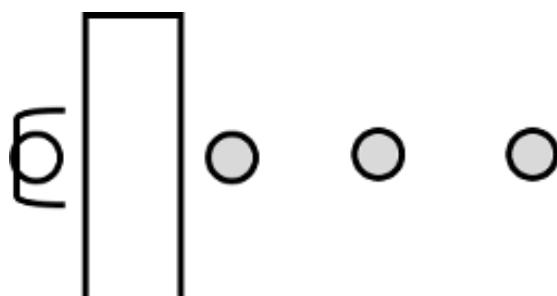


Figure 8.36.: Basic eight-foot table service station with 1 line

Finally, folding chairs will be used for patients to wait in the pre-vaccination area as well as the post-vaccination area. The chairs must be placed at least 6 feet apart. Figure 8.37 illustrates 12 folding chairs arranged for patient waiting. The number of chairs and how they are arranged are important design considerations. The company would like recommendations about size and arrangement of the pre-vaccination and post vaccination areas to ensure patient safety, utilization of the area, and patient flow. Clearly, a number of different layouts based on the total number of chairs is possible.

The placement and number of nurse/vaccination stations are of paramount importance to the operation of the clinic. The basic vaccination station has two chairs (one for the nurse and one for the patient) and a table for the nurse to utilize during the vaccination process. A nurse is dedicated to vaccinating the patient in a particular chair. The company is willing to consider a number of design configurations.

For a given set of nurse/vaccination stations, pharmacists are available to support the nurses. The pharmacist is needed to take the vaccine out of storage, load the hyper-dermic needle, and provide the needle and supplies to the nurse. In Figure 8.38, there is one pharmacist that is supporting two nurse/vaccination stations. In general, pharmacists prepare the vaccination dosages that the nurses inject into the patients. As shown in the figure, pharmacists have

8. Applications of Simulation Modeling

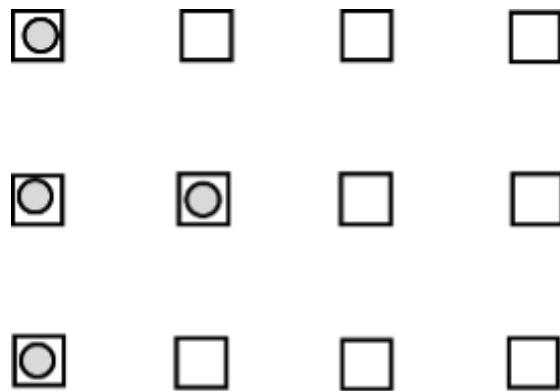


Figure 8.37.: Waiting area with 12 folding chairs and 4 seated patients

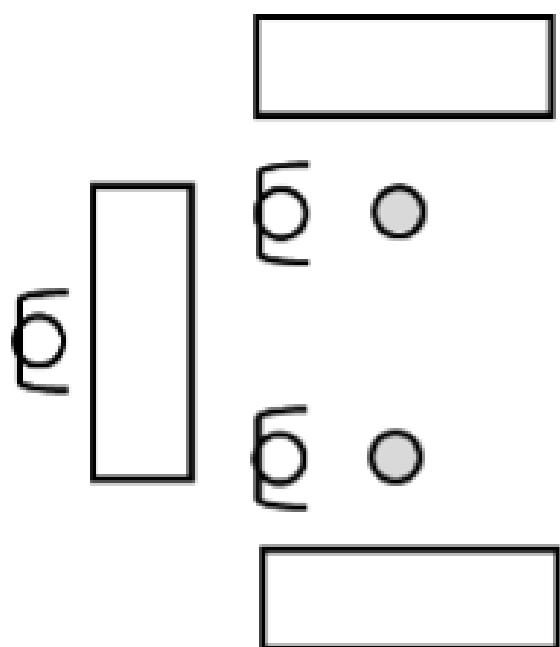


Figure 8.38.: Two nurse/vaccination stations supported by one pharmacist

their own work area for performing this task. There can be one large work area with many pharmacists that support all nursing stations or multiple pharmacy stations assigned to support specific nursing stations. Because the number of patients on the schedule is known at the beginning of the day, the pharmacists will start preparing vaccine dosages for use at the nurse/vaccination stations when the clinic first opens and continue to prepare dosages throughout the day. The pharmacists prepare the dosages based on the number of patients scheduled for the day, regardless of whether or not the patient actually is processed within the clinic. In this manner, an inventory of ready-made dosages is built up in the pharmacy area for use throughout the day. Whenever four dosages are made, the dosages are transported

to the nurse/vaccination station for use on patients. You can assume that there is always a volunteer available to move the dosages and that the time is based on the distance between the stations.

At the nurse/vaccination stations, the supplied vaccination doses are kept on hand, ready to use on patients. For simplicity, you can assume that the doses located within the nurse/vaccination area are close enough that any time to get a dose to a nurse at the station is negligible. If the quantity of doses at the nurse/vaccination station runs out, then the patients requiring a vaccination must wait until the inventory at the nurses/vaccination station is restocked. That is, patients cannot be vaccinated unless there is a dose ready for them.

Observation times for the pharmacist to fill a dose of the vaccine are available in a spreadsheet that accompanies this project. Unfortunately, during the operation of the clinic, incidents may arise that require a pharmacist's attention such that a pharmacist is needed to resolve the situation. This happens rarely, but during this time the pharmacist that handles the incident is no longer available to perform the nurse support activities. You can assume that during a 6-hour period, there will be on average 2 incidents, Poisson distributed randomly occurring within the interval. The time to handle the incident is quite variable and has a mean of 20 minutes, exponentially distributed. Any of the available pharmacists can handle the incident and a pharmacist will complete their current dosage making activity before handling the incident.

A concern for the design is how many pharmacists should support a set of nurse/vaccination stations. That is, should there be 1 for every two stations or 2 for 6 stations, or other pharmacist to nurse ratios or arrangements?

The company is interested in determining the number of basic nurse/vaccination stations, layout, and support pharmacists that are necessary to handle various throughput requirements. An example layout of the key clinic elements is illustrated in Figure @fig(fig:CovidProjectSystemOverview). In the diagram, there is one station for check-in, temperature checking, pre-vaccination, pre-vaccination seating, vaccination, and post vaccination processing. The company would like a more realistic layout that accommodates the 600-person throughput scenario.

A major constraint for the design of the clinic is the amount of square footage and how it is allocated to ensure that the social distancing requirements are met. In general, patients waiting for service during any part of the process must be able to maintain a 6-foot distance between themselves and other patients. In addition, workers that provide service to patients must be placed to ensure at least 6 feet of distance between each other. Eight-foot tables will be used to facilitate the placement of workers that are allocated to the check-in process, temperature checking, pre-vaccination, and post-vaccination checkout. In addition, small 1 foot tape dots can be placed within the facility to indicate locations where patients are permitted to stand. You may assume that you have up to 9000 square-feet of space for any design configurations you recommend; however, since rented space to operate the clinic can be expensive, you should try to minimize the amount of space (in square-feet) for your designs, while ensuring that you do not exceed 9000 square-feet of space. You should specify as part of your design approximately how

8. Applications of Simulation Modeling

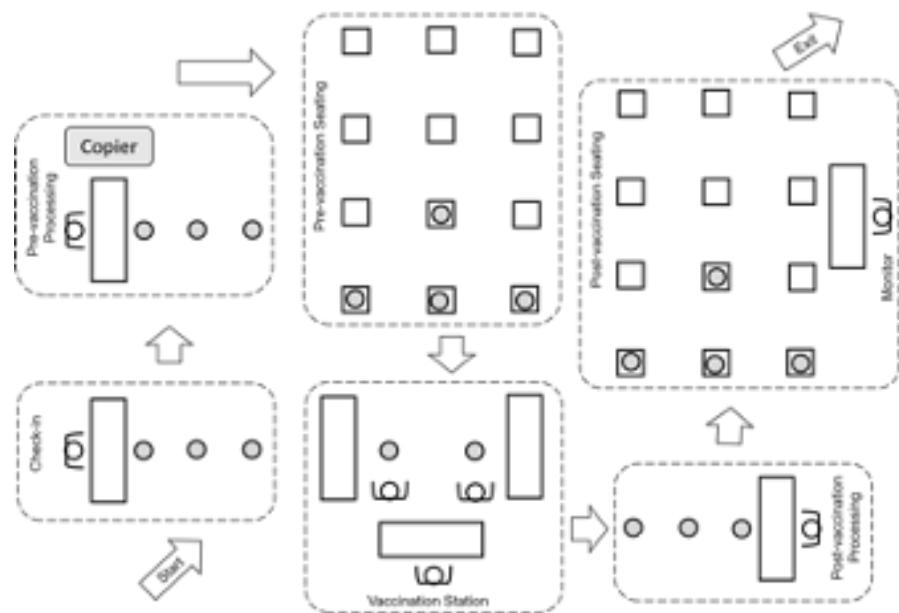


Figure 8.39.: Simplified diagram with key clinic elements

many square feet should be dedicated to the major areas of the clinic (check-in, pre-vaccination, pre-vaccination waiting, vaccination, post-vaccination, and post-vaccination seating).

Given the company's experience with providing flu vaccinations, they have acquired some data about the various processes; however, in some cases, they have only estimated values from domain experts. They are especially interested in understanding how the quality of the data might affect the operating projections for the clinics. The available data is summarized as follows:

- Even though patients may arrive at any time within a scheduling block, they tend to arrive near the start of the block, with separation between patients primarily due to walking from their cars. That is, not all patients scheduled for a particular block arrive to the clinic at exactly the start of the block. For simplicity, assume that the average distance from the parking lot to the clinic is approximately 50 yards.
- The speed of people walking can vary greatly, but prior data suggests that for short distances within and around buildings with social distancing, people walk between 1 miles per hour and 3 miles per hour, with a most likely time of 2 miles per hour.
- Because this is a new vaccine, data on no shows (people who schedule but fail to show up for the appointment) is not readily available. However, no show rates for general appointments at healthcare clinics can be significant, ranging between 12% to 24%. While the company does not think that no shows will be that high for a vaccine, there is some concern about how no shows may affect the operation of the clinic. For simplicity, you

should assume that there is a small chance of a no show, approximately, 1%, but the sensitivity to this assumption should be checked.

- A very small percentage (0.5%) of patients try to enter the facility without the required items. The requirements are explained and those that don't have the items need to leave. It is estimated that this check takes between 2 to 4 seconds.
- Under the assumption that a person having a high temperature is due to COVID, the chance that someone does not pass the temperature screening can be estimated based on the per person probability of infection. For example, if there are 50,000 new cases in a state of 10 million, the per-person probability is 0.5 percent. Because this can vary significantly by region and many other factors, this variable should be an input parameter for the model. For simplicity assume that 0.5% of the arriving patients will not pass the temperature screening. The temperature check takes between 5 and 15 seconds, uniformly distributed.
- Approximately, 2% of the arriving patients will not have their paperwork in order. It is estimated that it takes between 45 and 90 seconds uniformly distributed to use a photocopier to ensure paperwork compliance.
- Data associated with the pre-vaccination, vaccination and check-out processes is available in the provided spreadsheet for analysis and use within the modeling.
- Allergic reaction rates to the new vaccine are just beginning to be estimated but were extremely rare within the clinical trials. The chance of an allergic reaction should also be an input parameter so that sensitivity analysis can be performed. For simplicity, assume that initial data indicates that 0.002% of the post-vaccination patients will have an allergic reaction to the vaccine.

The patient schedule system is proprietary, and it has been difficult to get data on such a new process. Thus, some rough approximations will be required to model the demand for scheduling slots. The number of people expected to schedule an appointment on a given day is quite variable and is thought to be Poisson with a mean rate that will vary based on the population served. While the company would like design recommendations for different arrival rates, they specifically want a configuration for the case of a mean demand rate of 600 people per day. Data on scheduling clinics indicates that there are two types of scheduling preferences: 1) those people that have very flexible schedules and can pick any of the available slots at random and 2) those people that prefer slots that are later in the day. The break down between the two types of preferences really depends on the demographics of the underlying population (e.g. retirees tend to have more flexible schedules). For simplicity, you can assume that 60% of the people prefer late time windows and the remaining 40% have no preference. For those people that prefer late time windows, the general pattern has been as shown in Table 1.

Time period (hour of the day)	Chance of selection
1	5%
2	5%

8. Applications of Simulation Modeling

Time period (hour of the day)	Chance of selection
3	10%
4	20%
5	30%
6	30%

There was no data available on finer time periods. For simplicity, you can assume that within any of the hourly periods, finer division of the time periods would be equally likely to be chosen.

While the clinic is scheduled to be open for a fixed time period, e. g. 6 hours, all patients that enter the clinic during its operation must be processed. In addition, the company is committed to serving all patients that desire a vaccine on a given day. Thus, no limit is placed on the number of people that can be scheduled within a block; however, the company would like this monitored closely and accounted for within any design or staffing plan.

The company is interested in understanding the statistical properties of the following metrics:

- The probability that the clinic has to process patients after the scheduled operating hours.
- The number of patients in the clinic at the end of its scheduled operating length.
- The average time that patients spend waiting for each process.
- The total time that a patient spends in the clinic and how that time is broken down into the various waiting and activities.
- The utilization of the vaccination nurses.
- The utilization of the pharmacists.
- The utilization of the workers assigned to check-in, check-out, and pre-vaccination.
- The number of patients waiting to be vaccinated.
- The number of patients waiting in the post-vaccination area.
- The probability that arriving patients cannot enter the building due to inadequate social distancing space or bottlenecks within the process.
- The average waiting time of patients based on the hour of the day
- The utilization of the staff by hour of the day.

The company would like to know how to staff each area for demand rates of 550, 600, 650, 700 patients per day. That is, how many vaccination stations should they have and how many workers should be allocated to check-in, temperature checking, pre-vaccination, and post-vaccination. They would also like to determine the size of the waiting areas (i.e. the number of seats) and the square footage required for each processing area based on social distancing requirements. The

general goal is to have 80% of the patients experience a total vaccination experience time of less than 35 minutes. In addition, they would like to have 70% of patients in any hour of operation experience a total vaccination experience time of less than 35 minutes.

There has been some debate about how to organize and manage the waiting lines within the clinic and pre-vaccination waiting area. In particular, should individual lines be allocated to each nurse at each station or should a single line be used to feed multiple nurses that operate in parallel? For the pre-vaccination waiting area, the company is interested in recommendations that allow for fair and effective movement of patients waiting for a vaccination nurse. How will patients waiting within a general seating area know when it is their turn to proceed for a vaccination? The walking of patients within the facility should be accounted for within your solution.

While the staffing recommendations are of paramount importance, the company would like to know if changing the scheduling windows would be worthwhile. In addition, given the possibility of no shows, the company would like to understand how much over scheduling they should permit. That is, if the clinic is designed for mean demand of 600 people per day, how high can the rate go before the design needs to be changed? For a base configuration, the company is interested in an animation of the layout in order to assess the validity of the model and be used to educate personnel that need to manage the clinics.

A. Generating Pseudo-Random Numbers and Random Variates

LEARNING OBJECTIVES

- To be able to describe and use linear congruential pseudo-random number generation methods
- To be aware of current state of the art pseudo-random number generation methods
- To be able to define and use key terms in pseudo-random number generation methods such as streams, seeds, period, etc.
- To be able to explain the key issues in pseudo-random number testing
- To be able to derive and implement an inverse cumulative distribution function 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

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 section, 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. Then, given a good source of randomness, techniques have been established that permit the sequences to be transformed so that they can represent a wide variety of random variables (e.g. normal, Poisson, etc.). The algorithms that govern these procedures are described in the second part of this chapter.

A.1. Pseudo Random Numbers

This section indicates how uniformly distributed random numbers over the range from 0 to 1 are obtained within simulation programs. While commercial simulation packages provide

A. Generating Pseudo-Random Numbers and Random Variates

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 implement an algorithm to generate the random variates.

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

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 will not be truly random. For this reason, the random numbers that are used in computer simulation are called *pseudo random*.

Definition A.1 (Pseudo-Random Numbers). A sequence of pseudo-random 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 pseudo-random 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 pseudo-random numbers are called *random number generators*. In addition to passing a battery of statistical 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 such texts as (Fishman, 2006) and (Devroye, 1986).

A.1.1. Random Number Generators

Over the history of scientific computing, there have been a wide variety of techniques and algorithms proposed and used for generating pseudo-random numbers. A common technique that has been used (and is still in use) within a number of simulation environments is discussed in this text. Some new types of generators that have been recently adopted within many simulation environments, especially the one used within the JSL¹ and Arena, will also be briefly discussed.

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

Definition A.2 (Linear Congruential Generator). A LCG defines a sequence of integers, R_0, R_1, \dots between 0 and $m - 1$ according to the following recursive relationship:

$$R_{i+1} = (aR_i + c) \bmod m$$

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 a corresponding pseudo-random uniform number, we use

$$U_i = \frac{R_i}{m}$$

Notice that an LCG defines a sequence of integers and subsequently a sequence of real (rational) numbers that can be considered pseudo random numbers. Remember that pseudo random numbers are those that can “fool” a battery of statistical tests. The choice of the seed, constant multiplier, increment, and modulus, i.e. 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 pseudo random numbers. To make this concrete, let’s look at a simple example of an LCG.

¹<https://rossetti.git-pages.uark.edu/jslbookdownbook/>

A. Generating Pseudo-Random Numbers and Random Variates

Example A.1 (Simple LCG Example). Consider an LCG with the following 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's first remember how to compute using the mod operator. The mod operator is defined as:

$$z = y \bmod m = y - m \left\lfloor \frac{y}{m} \right\rfloor$$

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

$$\begin{aligned} z &= 17 \bmod 3 \\ &= 17 - 3 \left\lfloor \frac{17}{3} \right\rfloor \\ &= 17 - 3 \lfloor 5.66 \rfloor \\ &= 17 - 3 \times 5 = 2 \end{aligned} \tag{A.1}$$

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, the pseudo-random 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 the previous example, 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 a 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 pseudo random number properties to be obtained. The following result due to (Hull and Dobell, 1962), see also (Law, 2007), indicates how to check if a LCG will have the largest possible cycle.

Theorem A.1 (LCG Theorem). *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), and (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's apply this theorem to the example LCG and check whether or not it should obtain 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.



The theorem only tells us when a specification of (a, c, m) will obtain full period. In case (3), it does not tell us anything about what happens if 4 does not divide m . If 4 divides m , and we pick a so that it divides $(a - 1)$ then the condition will be met. If 4 does not divide m , then the theorem really does not tell us one way or the other whether the LCG obtains full period.

A. Generating Pseudo-Random Numbers and Random Variates

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 of m a prime number and $c = 0$, defines a special case of the LCG called a *prime modulus multiplicative linear congruential generator* (PMMLCG). 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 many simulation packages for a number of years. Notice that for PMMLCG's 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 A.1.

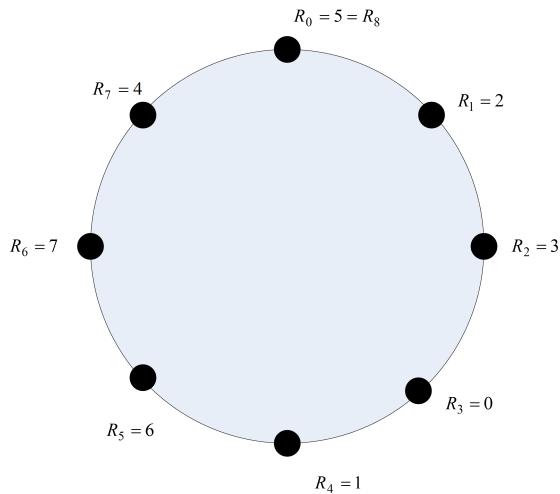


Figure A.1.: Sequence for Simple LCG Example

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 A.3 (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 subsequence stream 1 and the second subsequence stream 2, and so forth. Each stream can be further divided into subsequences or sub-streams of non-overlapping 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, e.g. $R_0 = 123098345$. It is difficult to remember such large numbers. Rather than remember 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 non-overlapping 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 non-overlapping 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

A. Generating Pseudo-Random Numbers and Random Variates

same pseudo random 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 generation of random number generators was developed that have extremely long periods. 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 3.1×10^{57} . This is the same generator that is now used in many commercial simulation packages. The generator as defined in (Law, 2007) is:

$$\begin{aligned} R_{1,i} &= (1,403,580R_{1,i-2} - 810,728R_{1,i-3})[\text{mod}(2^{32} - 209)] \\ R_{2,i} &= (527,612R_{2,i-1} - 1,370,589R_{2,i-3})[\text{mod}(2^{32} - 22,853)] \\ Y_i &= (R_{1,i} - R_{2,i})[\text{mod}(2^{32} - 209)] \\ U_i &= \frac{Y_i}{2^{32} - 209} \end{aligned}$$

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. To produce five pseudo random numbers using this generator we need an initial seed vector, such as:
 $\{R_{1,0}, R_{1,1}, R_{1,2}, R_{2,0}, R_{2,1}, R_{2,2}\} = \{12345, 12345, 12345, 12345, 12345, 12345\}$

Using the recursive equations, the resulting random numbers are as follows:

	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 sub-streams.



A random number stream is a sub-sequence of pseudo-random numbers that start at particular place with a larger sequence of pseudo-random numbers. The starting point of a sequence of pseudo-random numbers is called the *seed*. A seed allows us to pick a particular stream. Having multiple streams is useful to assign different streams to different sources of randomness within a model. Streams can be further divided into sub-streams. This facilitates the control of the use of pseudo-random numbers when performing experiments.

The fantastic thing about this generator is the sheer size of the period. Based on their analysis, L'Ecuyer et al. (2002) state 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 1.8×10^{19} with stream lengths of 1.7×10^{38} and sub-streams of length 7.6×10^{22} numbering at 2.3×10^{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. The subject of modeling and testing different distributions is deferred to a separate part of this book.

A.2. Generating Random Variates from Distributions

In simulation, pseudo random 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. Different distributions may require different algorithms due to the challenges of efficiently producing the random variables. Therefore, we need to know how to 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 cumulative distribution function (CDF) method
2. Convolution
3. Acceptance/Rejection

A. Generating Pseudo-Random Numbers and Random Variates

4. Mixture and Truncated Distributions

The following sections discuss each of these methods.

A.2.1. Inverse Transform Method

The inverse transform method is the preferred method for generating random variates provided that the inverse transform of the cumulative distribution function can be easily derived or computed numerically. The key advantage for the inverse transform method is that for every U_i use a corresponding X_i will be generated. That is, there is a one-to-one mapping between the pseudo-random number u_i and the generated variate x_i .

The inverse transform technique utilizes the inverse of the cumulative distribution function as illustrated in Figure A.2, will illustrates simple cumulative distribution function. First, generate a number, u_i between 0 and 1 (along the U axis), 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.

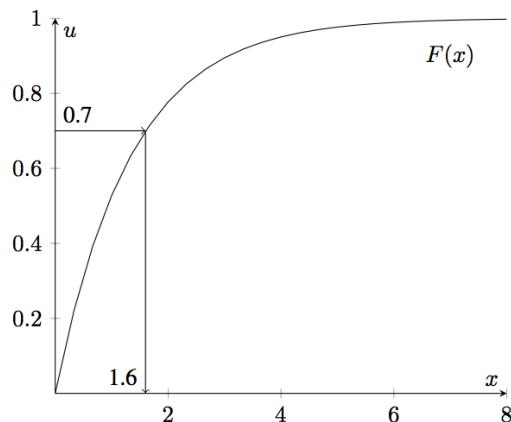


Figure A.2.: Inverse Transform Method

The idea illustrated in Figure A.2 is based on the following theorem.

Theorem A.2 (Inverse Transform). *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 this manner, then $Y \sim U(0, 1)$.*

The proof utilizes the definition of the cumulative distribution function 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.

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 cumulative distribution function to transform the random number to the appropriately distributed random variate.

Let's assume that we have a function, `randU01()`, that will provide pseudo-random numbers on the range (0,1). Then, the following presents the pseudo-code for the inverse transform algorithm.

1. $u = \text{rand01}()$
2. $x = F^{-1}(u)$
3. return x

Line 1 generates a uniform number. Line 2 takes the inverse of u and line 3 returns the random variate. The following example illustrates the inverse transform method for the exponential distribution.

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

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

with

$$\begin{aligned}
E[X] &= \frac{1}{\lambda} \\
Var[X] &= \frac{1}{\lambda^2}
\end{aligned}$$

A. Generating Pseudo-Random Numbers and Random Variates

Example A.2 (Generating Exponential Random Variates). Consider a random variable, X , that represents the time until failure for a machine tool. Suppose X is exponentially distributed with an expected value of 1.33 . Generate a random variate for the time until the first failure using a uniformly distributed value of $u = 0.7$.

Solution for Example A.2 In order to solve this problem, we must first compute the CDF for the exponential distribution. For any value, $b < 0$, we have by definition:

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

For any value $b \geq 0$,

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

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 $x = F^{-1}(u)$.

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

For Example A.2, we have that $E[X] = 1.33$. Since $E[X] = 1/\lambda$ for the exponential distribution, we have that $\lambda = 0.75$. Since $u = 0.7$, then the generated random variate, x , would be:

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

Thus, if we let $\theta = E[X]$, the formula for generating an exponential random variate is simply:

$$x = \frac{-1}{\lambda} \ln(1 - u) = -\theta \ln(1 - u) \quad (\text{A.2})$$

In the following pseudo-code, we assume that `randU01()` is a function that returns a uniformly distributed random number over the range (0,1).

1. $u = \text{randU01}()$
2. $x = \frac{-1}{\lambda} \ln(1 - u)$
3. return x

Thus, the key to applying the inverse transform technique for generating random variates is to be able to first derive the cumulative distribution function (CDF) and then to derive its inverse function. It turns out that for many common distributions, the CDF and inverse CDF well known.

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:

$$\begin{aligned} X &\sim \text{Uniform}(a, b) \\ f(x) &= \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \\ E[X] &= \frac{a+b}{2} \\ \text{Var}[X] &= \frac{(b-a)^2}{12} \\ F(x) &= \begin{cases} 0.0 & x < a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ 1.0 & x > b \end{cases} \end{aligned}$$

Example A.3 (Inverse CDF for Uniform Distribution). Consider a random variable, X , that represents the amount of grass clippings in a mower bag in pounds. Suppose the random variable is uniformly distributed between 5 and 35 pounds. Generate a random variate for the weight using a pseudo-random number of $u = 0.25$.

A. Generating Pseudo-Random Numbers and Random Variates

Solution for Example A.3 To solve this problem, we must determine the inverse CDF algorithm for the $U(a, b)$ distribution. The inverse of the CDF can be derived by setting $u = F(x)$ and solving for $x = F^{-1}(u)$.

$$\begin{aligned} u &= \frac{x - a}{b - a} \\ u(b - a) &= x - a \\ x &= a + u(b - a) = F^{-1}(u) \end{aligned}$$

For the example, we have that $a = 5$ and $b = 35$ and $u = 0.25$, 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 $F(x)$.

1. $u = \text{randU01}()$
2. $x = a + u(b - a)$
3. return x

For the previous distributions a closed form representation of the cumulative distribution function was available. If the cumulative distribution function can be inverted, then the inverse transform method can be easily used to generate random variates from the distribution. If no closed form analytical formula is available for the inverse cumulative distribution function, then often we can resort to numerical methods to implement the function. For example, the normal distribution is an extremely useful distribution and numerical methods have been devised to provide its inverse cumulative distribution function.

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 cumulative distribution function 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 cumulative distribution function can be searched to find the value of x associated with the given u . This process is illustrated in the following example.

Example A.4 (Discrete Empirical Distribution). Suppose you have a random variable, X , with the following discrete probability mass function and cumulative distribution function.

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

Plot the probability mass function and cumulative distribution function for this random variable. Then, develop an inverse cumulative distribution function for generating from this distribution. Finally, given $u_1 = 0.934$ and $u_2 = 0.1582$ are pseudo-random numbers, generate the two corresponding random variates from this PMF.

Solution for Example A.4 To solve this example, we must understand the functional form of the PMF and CDF, which are given as follows:

$$P\{X = x\} = \begin{cases} 0.4 & x = 1 \\ 0.3 & x = 2 \\ 0.2 & x = 3 \\ 0.1 & x = 4 \end{cases}$$

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

Figure A.3 illustrates the CDF for this discrete distribution.

Examining Figure A.3 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 the inverse of the cumulative distribution function.

A. Generating Pseudo-Random Numbers and Random Variates

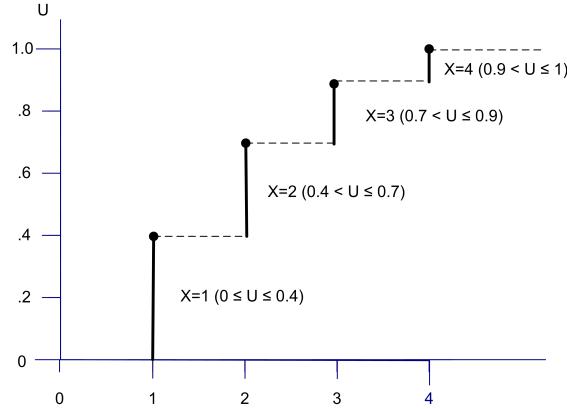


Figure A.3.: Example Empirical CDF

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

Suppose $u_1 = 0.934$, then by $F^{-1}(u)$, $x = 4$. If $u_2 = 0.1582$, then $x = 1$. Thus, we use the inverse transform function 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 table lookup 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. The following presents these ideas in the form of an algorithm.

1. $u = \text{randU010}$
2. $i = 1$
3. $x = x_i$
4. WHILE $F(x) \leq u$
5. $i = i + 1$
6. $x = x_i$
7. END WHILE
8. RETURN x

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

1. $u = \text{randu01()}$
2. IF ($u \leq p$) 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 the following algorithm:

1. $u = \text{randU01}()$
2. $x = a + \lfloor (b - a + 1)u \rfloor$
3. return x

Notice how the discrete uniform distribution inverse transform algorithm is different from the case of the continuous uniform distribution associated with Example A.4. 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:

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

We will call this the shifted geometric in this text. The algorithm to generate a shifted geometric random variables is as follows:

1. $u = \text{randU01}()$
2. $x = 1 + \left\lfloor \frac{\ln(1-u)}{\ln(1-p)} \right\rfloor$
3. return x

Notice the use of the floor operator $\lfloor \cdot \rfloor$. 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:

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

We will call this distribution the geometric distribution in this text. The algorithm to generate a geometric random variables is as follows:

A. Generating Pseudo-Random Numbers and Random Variates

1. $u = \text{randU01}()$
2. $y = \left\lfloor \frac{\ln(1-u)}{\ln(1-p)} \right\rfloor$
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 we have an interval of real numbers, and that incidents occur at random throughout the interval. If the interval can be partitioned into sub-intervals of small enough length such that:

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

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 λ remains constant. To keep λ 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:

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

where $E[X] = \lambda$ and $Var[X] = \lambda$. 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 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 $E[X(t)] = \lambda t$. 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 $E[X(t)] = \lambda t$. Therefore, a method for generating Poisson random variates with mean λ 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 A.5 (Generate Poisson Random Variates). 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 the following pseudo-random number (0.971, 0.687, 0.314, 0.752, 0.830) to generate a value of $X(2)$. That is, generate the number of arrivals in 2 hours.

A. Generating Pseudo-Random Numbers and Random Variates

Solution for Example A.5 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 provided u_i , we can compute T_i (via the inverse transform method for the exponential distribution) 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 time 2 hours, $X(2) = 4$. That is, there were 4 customers that arrived within the 2 hours. This example is meant to be illustrative of one method for generating Poisson random variates. There are much more efficient methods that have been developed.

The inverse transform technique is general and is used when $F^{-1}(\cdot)$ is 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.

A.2.2. 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 independent and identically distributed 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. Let's look at a couple of examples. By definition, a negative binomial distribution represents one of the following two random variables:

- The number of failures in sequence of Bernoulli trials before the r^{th} success, has range $\{0, 1, 2, \dots\}$.
- The number of trials in a sequence of Bernoulli trials until the r^{th} success, it has range $\{r, r + 1, r + 2, \dots\}$

The number of failures before the r^{th} success, has range $\{0, 1, 2, \dots\}$. This is the sum of geometric random variables with range $\{0, 1, 2, \dots\}$ with the same success probability.

If $Y \sim NB(r, p)$ with range $\{0, 1, 2, \dots\}$, then

$$Y = \sum_{i=1}^r X_i$$

when $X_i \sim \text{Geometric}(p)$ with range $\{0, 1, 2, \dots\}$, and X_i can be generated via inverse transform with:

$$X_i = \left\lfloor \frac{\ln(1 - U_i)}{\ln(1 - p)} \right\rfloor$$

Note that $\lfloor \cdot \rfloor$ is the floor function².

If we have a negative binomial distribution that represents the number of trials until the r^{th} success, it has range $\{r, r + 1, r + 2, \dots\}$, in this text we call this a shifted negative binomial distribution.

A random variable from a “shifted” negative binomial distribution is the sum of shifted geometric random variables with range $\{1, 2, 3, \dots\}$ with same success probability. In this text, we refer to this geometric distribution as the shifted geometric distribution.

If $T \sim NB(r, p)$ with range $\{r, r + 1, r + 2, \dots\}$, then

²https://en.wikipedia.org/wiki/Floor_and_ceiling_functions#:~:text=In%20the%20language%20of%20order,tegers%20into%20the%20reals.

A. Generating Pseudo-Random Numbers and Random Variates

$$T = \sum_{i=1}^r X_i$$

when $X_i \sim \text{Shifted Geometric}(p)$ with range $\{1, 2, 3, \dots\}$, and X_i can be generated via inverse transform with:

$$X_i = 1 + \left\lfloor \frac{\ln(1 - U_i)}{\ln(1 - p)} \right\rfloor$$

Notice that the relationship between these random variables as follows:

Let Y be the number of failures in a sequence of Bernoulli trials before the r^{th} success.

Let T be the number of trials in a sequence of Bernoulli trials until the r^{th} success,

Then, clearly, $T = Y + r$. Notice that we can generate Y , via convolution, as previously explained and just add r to get T .

$$Y = \sum_{i=1}^r X_i$$

Where $X_i \sim \text{Geometric}(p)$ with range $\{0, 1, 2, \dots\}$, and X_i can be generated via inverse transform with:

$$X_i = \left\lfloor \frac{\ln(1 - U_i)}{\ln(1 - p)} \right\rfloor \quad (\text{A.3})$$

Example A.6 (Generate Negative Binomial Variates via Convolution). Use the following pseudo-random numbers $u_1 = 0.35$, $u_2 = 0.64$, $u_3 = 0.14$, generate a random variate from Negative Binomial distribution having parameters $r = 3$ and $p = 0.3$. Also, using the same pseudo-random numbers, generate a random variate from a *shifted* Negative Binomial distribution having parameters $r = 3$ and $p = 0.3$.

Solution for Example A.6 To solve this example, we need to apply (A.3) to each provided u_i to compute the three values of X_i .

$$\begin{aligned} X_1 &= \left\lfloor \frac{\ln(1 - 0.35)}{\ln(1 - 0.3)} \right\rfloor = 1 \\ X_2 &= \left\lfloor \frac{\ln(1 - 0.64)}{\ln(1 - 0.3)} \right\rfloor = 2 \end{aligned}$$

$$X_3 = \left\lfloor \frac{\ln(1 - 0.14)}{\ln(1 - 0.3)} \right\rfloor = 0$$

Thus, we have that $Y = X_1 + X_2 + X_3 = 1 + 2 + 0 = 3$. To generate T for the shifted negative binomial, we have that $T = Y + r = Y + 3 = 3 + 3 = 6$. Notice that this all can be easily done within a spreadsheet or within a computer program.

As another example of using convolution consider the requirement to generate random variables from an Erlang distribution. Suppose that $Y_i \sim \text{Exp}(E[Y_i] = 1/\lambda)$. That is, Y is exponentially distributed with rate parameter λ . Now, define X as $X = \sum_{i=1}^r Y_i$. One can show that X will have an Erlang distribution with parameters (r, λ) , where $E[X] = r/\lambda$ and $\text{Var}[X] = r/\lambda^2$. Thus, an Erlang(r, λ) is an r -fold convolution of r exponentially distributed random variables with common mean $1/\lambda$.

Example A.7 (Generate Erlang Random Variates via Convolution). Use the following pseudo-random numbers $u_1 = 0.35$, $u_2 = 0.64$, $u_3 = 0.14$, generate a random variate from an Erlang distribution having parameters $r = 3$ and $\lambda = 0.5$.

Solution for Example A.7 This requires generating 3 exponential distributed random variates each with $\lambda = 0.5$ and adding them up.

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

Because of its simplicity, the convolution method is easy to implement; however, in a number of cases (in particular for a large value of n), there are more efficient algorithms available.

A.2.3. 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, $w(x)$, that 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 $-\infty < x < +\infty$.

A. Generating Pseudo-Random Numbers and Random Variates

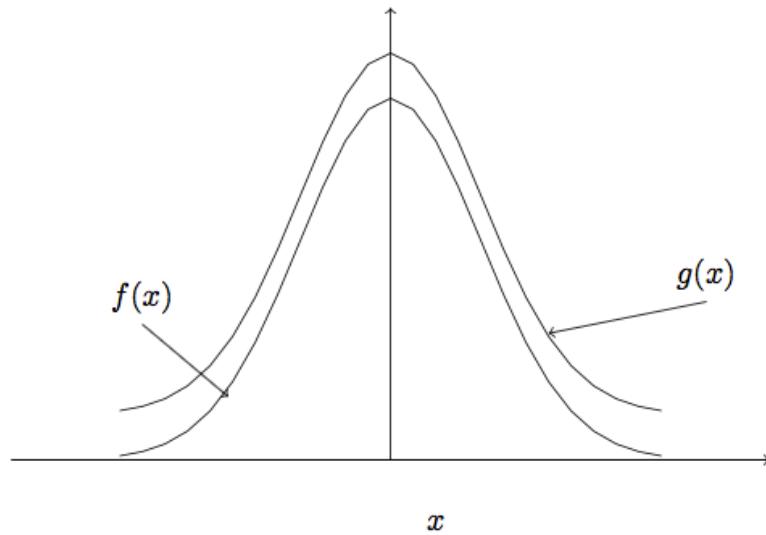


Figure A.4.: Concept of a Majorizing Function

Figure A.4 illustrates the concept of a majorizing function for $f(x)$, which simply means a function that is bigger than $f(x)$ everywhere.

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

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

If $w(x)$ is defined as $w(x) = g(x)/c$ then $w(x)$ will be a probability density function. 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, e.g. via the inverse transform method. Let $U \sim U(0, 1)$. The steps of the procedure are as provided in the following algorithm. 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.

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 cumulative distribution function of W given that the $W = w$ was accepted, $P\{W \leq x | W = w \text{ is accepted}\}$.

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 A.8 (Acceptance-Rejection Example). Consider the following PDF over the range $[-1, 1]$. Develop an acceptance/rejection based algorithm for $f(x)$.

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

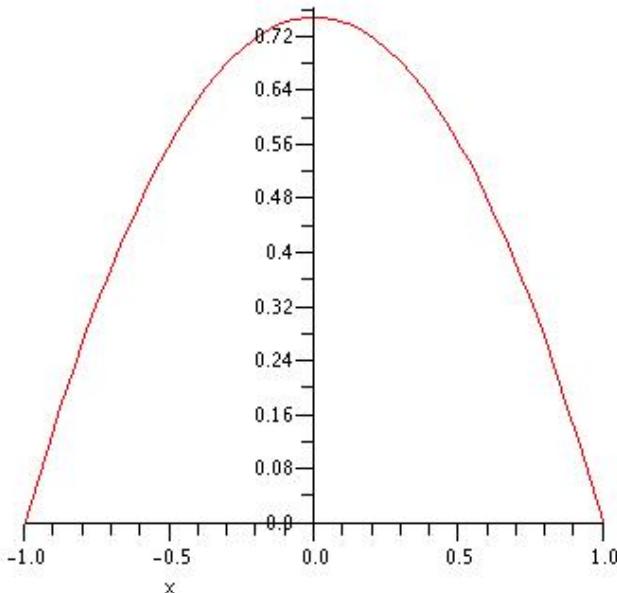


Figure A.5.: Plot of $f(x)$

Solution for Example A.8 The first step in deriving an acceptance/rejection algorithm for the $f(x)$ in Example A.8 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:

A. Generating Pseudo-Random Numbers and Random Variates

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

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 the discussion of the continuous uniform distribution, 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 as follows:

1. Repeat
 - 1.1 Generate $U_1 \sim U(0, 1)$
 - 1.2 $W = -1 + 2U_1$
 - 1.3 Generate $U_2 \sim U(0, 1)$
 - 1.4 $f = \frac{3}{4}(1 - W^2)$
2. Until $(U_2 \times \frac{3}{4} \leq f)$
3. Return W

Steps 1.1 and 1.2 of 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 1.4 is simply finding the height associated with W in terms of f . Step 1.3 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 2, 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 the previous, 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:

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

where c is the area under the majorizing function. For the example, 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 probability density function, $f(x)$, if it has an infinite range because it may be more challenging to find a good majorizing function.

A.2.4. 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's consider the definition of a mixture distribution and then consider some examples.

Definition A.4 (Mixture Distribution). The distribution of a random variable X is a *mixture distribution* if the CDF of X has the form:

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

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

Notice that the ω_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 $g(i) = \{I = i\} = \omega_i$ and then generate X from $F_{X_I}(x)$. The following algorithm presents this procedure.

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, lognormal, etc., have a single mode. Mixture distributions are often utilized for the modeling of data sets that have more than one mode.

A. Generating Pseudo-Random Numbers and Random Variates

As an example of a mixture distribution, we will discuss the hyper-exponential distribution. The hyper-exponential 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 $\sigma = \sqrt{Var[X]}$ and $\mu = E[X]$. For the hyper-exponential distribution $c_v > 1$. The hyper-exponential 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. The following example illustrates how to generate from a hyper-exponential distribution.

Example A.9 (Hyper-Exponential Random Variate). Suppose the time that it takes to pay with a credit card, X_1 , is exponentially distributed with a mean of 1.5 minutes and the time that it takes to pay with cash, X_2 , is exponentially distributed with a mean of 1.1 minutes. 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 hyper-exponential 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 pseudo-random numbers, $u_1 = 0.54$ and $u_2 = 0.12$, generate a random variate from $F_X(x)$.

Solution for Example A.9 In order to generate a payment service time, X , we can use the mixture distribution algorithm.

1. Generate $u \sim U(0, 1)$
2. Generate $v \sim U(0, 1)$
3. If ($u \leq 7$)
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 the previous example, 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, $F_{X_3}(x)$ might use convolution, etc. 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, the random variable X given that X is contained in $[a, b]$. 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:

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

where

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

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

A. Generating Pseudo-Random Numbers and Random Variates

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

Thus, $f^*(x)$ is simply a “re-weighting” of $f(x)$. The CDF of $f^*(x)$ is:

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

This leads to a straight forward algorithm for generating from $f^*(x)$ as follows:

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 a X given that $a \leq X \leq b$. Let’s look at an example.

Example A.10 (Generating a Truncated Random Variate). 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 pseudo-random number 0.23.

Solution for Example A.10 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 cumulative distribution function:

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

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, in order 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 δ .

Example A.11 (Generating a Shifted Weibull Random Variate Example). Suppose X represents the time to setup a machine for production. From past time studies, we know that it cannot take any 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 $\alpha = 3$ and scale parameter $\beta = 5$. Using a pseudo-random number of $u = 0.73$ generate a value for the time to perform the setup.

Solution for Example A.11 The Weibull distribution has a closed form cumulative distribution function:

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

Thus, the inverse CDF function is:

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

Therefore to generate the setup time we have:

$$5.5 + 5 [-\ln(1 - 0.73)]^{1/3} = 5.5 + 5.47 = 10.97$$

Within this section, we illustrated the four primary methods for generating random variates 1) inverse transform, 2) convolution, 3) acceptance/rejection, and 4) mixture and truncated distributions. These are only a starting point for the study of random variate generation methods.

A. Generating Pseudo-Random Numbers and Random Variates

A.3. Summary

This section covered a number of important concepts used within simulation including:

- Generating pseudo-random numbers
- Generating random variates and processes

Appendices F.1 and F.2 summarize many of the properties of common discrete and continuous distributions. These topics provide a solid foundation for modeling random components within simulation models. Not only should you now understand how random numbers are generated you also know how to transform those numbers to allow the generation from a wide variety of probability distributions. To further your study of random variate generation, you should study the generation of multi-variate distributions.

A.4. Exercises

Exercise A.1. The sequence of random numbers generated from a given seed is called a random number (a)_____.

Exercise A.2. State three major methods of generating random variables from any distribution.
 (a)_____. (b)_____. (c)_____.

Exercise A.3. Consider the multiplicative congruential generator with ($a = 13, m = 64, c = 0$, and seeds $X_0 = 1, 2, 3, 4$). a) Using Theorem A.1, does this generator achieve its maximum period for these parameters? b) Generate one period's worth of uniform random variables from each of the supplied seeds.

Exercise A.4. Consider the multiplicative congruential generator with ($a = 11, m = 64, c = 0$, and seeds $X_0 = 1, 2, 3, 4$). a) Using Theorem A.1, does this generator achieve its maximum period for these parameters? b) Generate one period's worth of uniform random variables from each of the supplied seeds.

Exercise A.5. Consider the linear congruential generator with ($a = 11, m = 16, c = 5$, and seed $X_0 = 1$). a) Using Theorem A.1, does this generator achieve its maximum period for these parameters? b) Generate 2 pseudo-random uniform numbers for this generator.

Exercise A.6. Consider the linear congruential generator with ($a = 13, m = 16, c = 13$, and seed $X_0 = 37$). a) Using Theorem A.1, does this generator achieve its maximum period for these parameters? b) Generate 2 pseudo-random uniform numbers for this generator.

A. Generating Pseudo-Random Numbers and Random Variates

Exercise A.7. Consider the linear congruential generator with ($a = 8$, $m = 10$, $c = 1$, and seed $X_0 = 11$). a) Using Theorem A.1, does this generator achieve its maximum period for these parameters? b) Generate 2 pseudo-random uniform numbers for this generator.

Exercise A.8. 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

- a. Determine the CDF $F(x)$ for the random variable, X .
 - b. Create a graphical summary of the CDF. See Example A.4.
 - c. Create a look-up table that can be used to determine a sample from the discrete distribution, $p(x)$. See Example A.4.
 - d. Generate 3 values of X using the following pseudo-random numbers $u_1 = 0.943$, $u_2 = 0.398$, $u_3 = 0.372$
-

Exercise A.9. 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 1st random number.
 - b. Generate a random variate from a (12, 22) discrete uniform distribution using the 2nd random number.
-

Exercise A.10. 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 1 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 mass function:

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 4 random variates from the probability mass function.

Exercise A.11. Suppose that customers arrive at an ATM via a Poisson process with mean 7 per hour. Determine the arrival time of the first 6 customers using the following pseudo-random numbers via the inverse transformation method. Start with the first row and read across the table.

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

Exercise A.12. The demand, D , for parts at a repair bench per day can be described by the following discrete probability mass function:

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

Generate the demand for the first 4 days using the following sequence of $U(0,1)$ random numbers: 0.943, 0.398, 0.372, 0.943.

A. Generating Pseudo-Random Numbers and Random Variates

Exercise A.13. The service times for a 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 $\lambda = 45$. Generate two service times for this distribution using the following sequence of U(0,1) random numbers: 0.943, 0.398, 0.372, 0.943.

Exercise A.14. 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. Generate two random variates from this distribution using the following sequence of U(0,1) random numbers: 0.943, 0.398, 0.372, 0.943.

Exercise A.15. The interest rate for a capital project is unknown. An accountant has estimated that the minimum interest rate will 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. Generate two random variates from this distribution using the following sequence of U(0,1) random numbers: 0.943, 0.398, 0.372, 0.943.

Exercise A.16. 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. Start with the first row and read across the table determine the arrival times of the first three customers at each server.

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

Exercise A.17. 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 0.943, 0.398, 0.372, 0.943, 0.204 generate 5 random variates from the triangular distribution with $a = 2$, $c = 5$, $b = 10$.
-

Exercise A.18. Consider the following probability density function:

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

- a. Derive an inverse transform algorithm for this distribution. b. Using 0.943, 0.398 generate two random variates from this distribution.
-

Exercise A.19. Consider the following probability density function:

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

- a. Derive an inverse transform algorithm for this distribution. b. Using 0.943, 0.398 generate two random variates from this distribution.
-

Exercise A.20. Consider the following probability density function:

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

A. Generating Pseudo-Random Numbers and Random Variates

- a. Derive an inverse transform algorithm for this distribution. b. Using 0.943, 0.398 generate two random variates from this distribution.
-

Exercise A.21. Consider the following probability density function:

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

- a. Derive an inverse transform algorithm for this distribution. b. Using 0.943, 0.398 generate two random variates from this distribution.
-

Exercise A.22. The times to failure for an automated production process have been found to be randomly distributed according to a Rayleigh distribution:

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

- a. Derive an inverse transform algorithm for this distribution. b. Using 0.943, 0.398 generate two random variates from this distribution with $\beta = 2.0$.
-

Exercise A.23. Using the first two rows of random numbers from the following table, generate 5 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 4 successes.

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

Exercise A.24. Using the first two rows of random numbers from the following table, generate 5 random numbers from the negative binomial distribution with parameters ($r = 4, p = 0.4$) using a sequence of Bernoulli trials to get 4 successes.

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

Exercise A.25. 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 the following table 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:

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

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

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

Exercise A.26. 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 the following table generate the service time for two patients.

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

Exercise A.27. If $Z \sim N(0, 1)$, and $Y = \sum_{i=1}^k Z_i^2$ then $Y \sim \chi_k^2$, where χ_k^2 is a chi-squared random variable with k degrees of freedom. Using the first two rows of random numbers from the following table generate two χ_5^2 random variates.

A. Generating Pseudo-Random Numbers and Random Variates

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

Exercise A.28. 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.

Exercise A.29. 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.

Exercise A.30. Consider the following probability density function:

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

- a. Derive an acceptance-rejection algorithm for this distribution. b. Using the first row of random numbers from the following table generate 2 random variates using your algorithm.

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

Exercise A.31. 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 α 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:

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

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

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

Exercise A.32. Parts arrive to a machine center with three drill presses according to a Poisson distribution with mean λ . 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 inter-arrival times to each drill press? Specify the parameters of the distribution. Suppose that p_1 , p_2 , and p_3 equal to 0.25, 0.45, and 0.3 respectively and that λ is equal to 12 per minute.

Using the first row of random numbers from the following table generate the first three arrival times.

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

Exercise A.33. Consider the following function:

$$f(x) = \begin{cases} cx^2 & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

- a. Determine the value of c that will turn $g(x)$ into a probability density function. The resulting probability density function is called a parabolic distribution.
 - b. Denote the probability density function found in part (a), $f(x)$. Let X be a random variable from $f(x)$. Derive the inverse cumulative distribution function for $f(x)$.
-

Exercise A.34. Consider the following probability density function:

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

Derive an inverse cumulative distribution algorithm for generating from $f(x)$.

B. Probability Distribution Modeling

LEARNING OBJECTIVES

- To be able model discrete distributions based on data
- To be able model continuous distributions based on data
- To be able to perform basic statistical tests on uniform pseudo-random numbers

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, but 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.

B. Probability Distribution 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 defining 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 analysis of the data: Using standard statistical analysis tools you should visually examine your data. This should include such plots as a histogram, a time series plot, and an auto-correlation plot. Again, using statistical analysis tools you should summarize the basic statistical properties of the data, e.g. sample average, sample variance, minimum, maximum, quartiles, etc.
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, Kolmogorov-Smirnov Test). As 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 build credibility and allows you to illustrate how the data was collected.

The following section provides a review of probability and statistical concepts that are useful in distribution modeling.

B.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

B.1. Random Variables and Probability Distributions

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. As shown in Figure B.1, 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 .

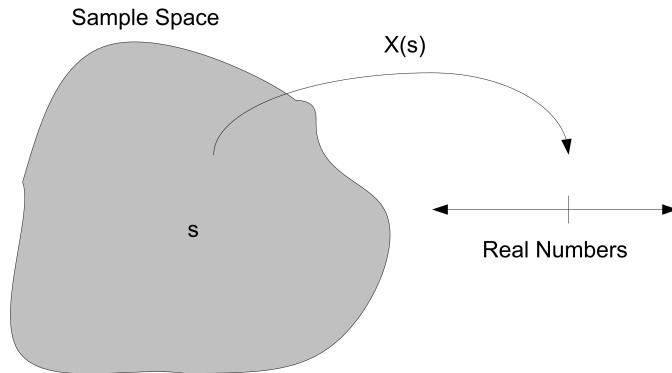


Figure B.1.: Random Variables Map Outcomes to Real Numbers

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, i.e. 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, i.e. $X_i \in \mathbb{R}^+$ or in other words, $X_i \geq 0$, then the service time should be modeled as a *continuous* random variable.

Suppose instead that the service time can only take on one of five discrete values 2, 5, 7, 8, 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. As indicated in Figure B.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:

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

where $f(x_i) \geq 0$ for all x_i and $\sum_{i=1}^n f(x_i) = 1$. The probability mass function describes the probability value associated with each discrete value of the random variable.

B. Probability Distribution Modeling

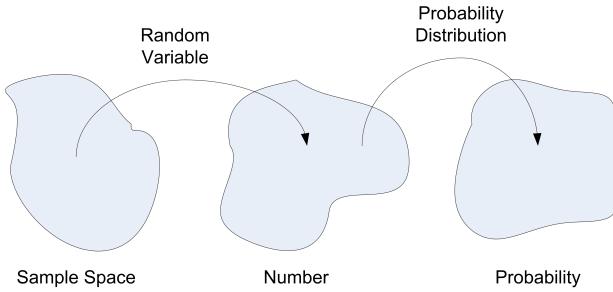


Figure B.2.: robability Distributions Map Random Variables to Probabilities

For a continuous random variable, X , the mapping from real numbers to probability values is governed by a probability density function, $f(x)$ and has the properties:

1. $f(x) \geq 0$
2. $\int_{-\infty}^{\infty} f(x)dx = 1$ (The area must sum to 1.)
3. $P(a \leq x \leq b) = \int_a^b f(x)dx$ (The area under $f(x)$ between a and b .)

The probability density function (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 cumulative distribution function is defined as

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

and satisfies, $0 \leq F(x) \leq 1$, and for $x \leq y$ then $F(x) \leq F(y)$.

The cumulative distribution function of a continuous random variable is

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

Thus, when modeling the elements of a simulation model that have randomness, one must determine:

- 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:

$$E[X] = \sum_x x f(x)$$

where the sum is defined through all possible values of x . The *variance* of X is denoted by $Var[X]$ and is defined as:

$$\begin{aligned} Var[X] &= E[(X - E[X])^2] \\ &= \sum_x (x - E[X])^2 f(x) \\ &= \sum_x x^2 f(x) - (E[X])^2 \\ &= E[X^2] - (E[X])^2 \end{aligned}$$

Suppose X is a continuous random variable with PDF, $f(x)$, then the expected value of X is

$$E[X] = \int_{-\infty}^{\infty} x f(x) dx$$

and the variance of X is

$$Var[X] = \int_{-\infty}^{\infty} (x - E[X])^2 f(x) dx = \int_{-\infty}^{\infty} x^2 f(x) dx - (E[X])^2$$

which is equivalent to $Var[X] = E[X^2] - (E[X])^2$ 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:

$$c_v = \frac{\sqrt{Var[X]}}{E[X]}$$

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, $\bar{X}(n)$,

B. Probability Distribution Modeling

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

is often used. To estimate $Var[X]$, assuming that the data are independent, the sample variance, S^2 ,

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

can be used. Thus, an estimator for the coefficient of variation is:

$$\hat{c}_v = \frac{s}{\bar{x}}$$

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 (1st quartile, median, 3rd quartile) - Summarizes the distribution of the data.
- minimum, maximum, and range - Indicates the range of possible values

Skewness can be estimated by:

$$\hat{\gamma}_1 = \frac{\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^3}{[S^2]^{3/2}}$$

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:

$$\hat{\gamma}_2 = \frac{n-1}{(n-2)(n-3)} ((n+1)g_2 + 6)$$

where, g_2 is:

$$g_2 = \frac{\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^4}{\left(\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2\right)^2} - 3$$

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:

$$\tilde{x} = \begin{cases} x_{((n+1)/2)} & n \text{ is odd} \\ \frac{x_{(n/2)} + x_{((n/2)+1)}}{2} & n \text{ is even} \end{cases}$$

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:

$$\frac{n+1}{2} = \frac{5+1}{2} = 3$$

$$\tilde{x} = x_{(3)} = 7$$

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.

B. Probability Distribution Modeling

As noted in this section, a key decision in distribution modeling is whether the underlying random variable is discrete or continuous. The next section discusses how to model discrete distributions.

B.2. Modeling with Discrete Distributions

There are a wide variety of discrete random variables that often occur in simulation modeling. Appendix F.1 summarizes the functions and characteristics some common discrete distributions. Table B.1 provides an overview of some modeling situations for common discrete distributions.

Table B.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(λ)	counts of occurrences in an interval, area, or volume

By understanding the modeling situations that produce data, you can hypothesize the appropriate distribution for the distribution fitting process.

B.3. 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 *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!).

B.3.1. 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 B.1 presents data collected from an arrival process. As noted in Table B.1, the Poisson distribution is a prime candidate for modeling this type of data.

Example B.1 (Fitting a Poisson Distribution). Suppose that we are interested in modeling the demand for a computer laboratory during the morning hours between 9 am to 11 am on normal weekdays. During this time a team of undergraduate students has collected the number of students arriving to the computer lab during 15 minute intervals over a period of 4 weeks.

Since there are four 15 minute intervals in each hour for each two hour time period, there are 8 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. A sample of observations per 15 minute interval are presented in Table B.2. The full data set is available with the chapter files. Check whether a Poisson distribution is an appropriate model for this data.

Table B.2.: 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

The solution to Example B.1 involves the following steps:

1. Visualize the data.

B. Probability Distribution Modeling

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.

B.3.2. 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")
```

Table B.3 illustrates the layout of the data frame in R. The first column indicates the week, the 2nd column represents the period of the day, the 3rd 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. Notice that each period of the day is labeled numerically. To access a particular column within the data frame you use the \$ operator. Thus, the reference, $p\$N$ accesses all the counts across all of the week, period, day combinations. The variable, $p\$N$, is subsequently used in the *hist*, *plot*, and *acf* commands.

As can be seen in Figure B.3 the data has a somewhat symmetric shape with nothing unusual appearing in the figure. The shape of the histogram is consistent with possible shapes associated with the Poisson distribution.

The time series plot, shown in Figure B.4, illustrates no significant patterns. We see random looking data centered around a common mean value with no trends of increasing or decreasing data points and no cyclical patterns of up and down behavior.

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:

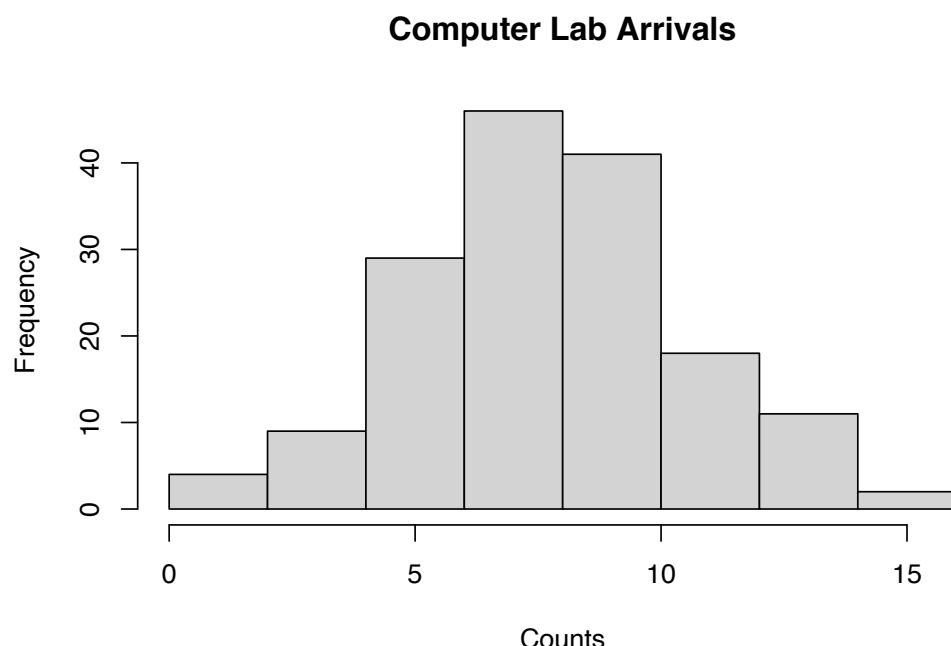


Figure B.3.: Histogram of Computer Lab Arrivals

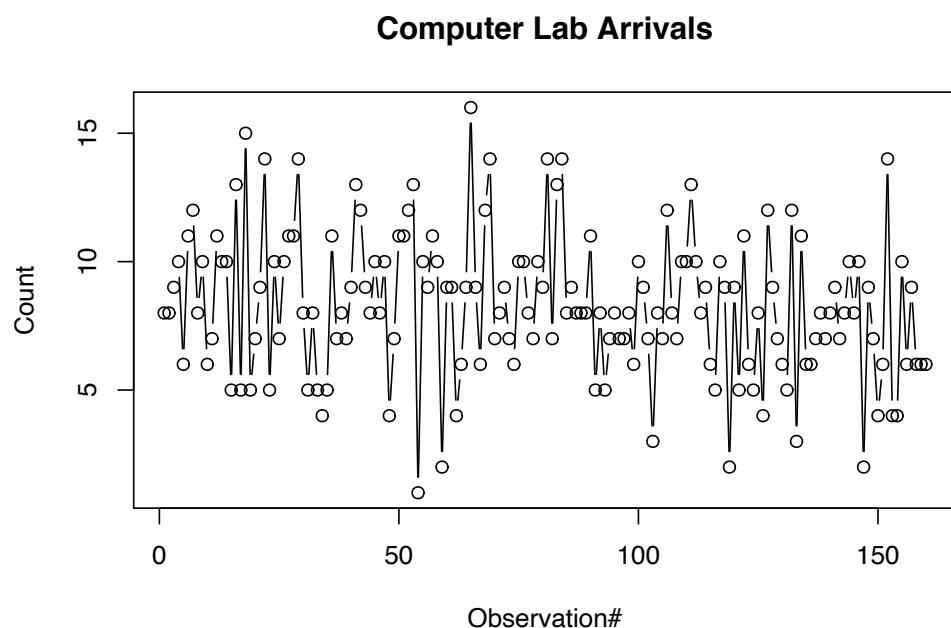


Figure B.4.: Time Series Plot of Computer Lab Arrivals

B. Probability Distribution Modeling

Table B.3.: Computer Lab Arrival Data.

Week	Period	Day	N
1	1	M	8
1	2	M	8
1	3	M	9
1	4	M	10
1	5	M	6
1	6	M	11
1	7	M	12
1	8	M	8
2	1	M	10
2	2	M	6
2	3	M	7
2	4	M	11
2	5	M	10
2	6	M	10
2	7	M	5
2	8	M	13
3	1	M	5
3	2	M	15
3	3	M	5
3	4	M	7

- 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, $\rho_k = cor[X_i, X_{i+k}]$, 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 upon where the points are in the series, it depends only upon the distance between them in the series.

Recall that the correlation between two random variables is defined as:

$$cor[X, Y] = \frac{cov[X, Y]}{\sqrt{var[X]Var[Y]}}$$

where the covariance, $cov[X, Y]$ is defined by:

$$\text{cov}[X, Y] = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y]$$

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:

$$\begin{aligned}\rho_k &= \text{cor}[X_i, X_{i+k}] = \frac{\text{cov}[X_i, X_{i+k}]}{\sqrt{\text{Var}[X_i]\text{Var}[X_{i+k}]}} \\ &= \frac{\text{cov}[X_i, X_{i+k}]}{\sigma^2} \text{ for } k = 1, 2, \dots\end{aligned}$$

A plot of ρ_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 ρ_k from the actual times series. This involves forming an estimator for ρ_k . (Law, 2007) suggests plotting:

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

where

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

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

$$\bar{X}(n) = \frac{1}{n} \sum_i^n X_i \quad (\text{B.3})$$

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

B. Probability Distribution Modeling

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

$$r_k = \frac{c_k}{c_0} = \frac{\sum_{i=1}^{n-k} (X_i - \bar{X}(n)) (X_{i+k} - \bar{X}(n))}{\sum_{i=1}^n (X_i - \bar{X}(n))^2} \quad (\text{B.4})$$

where

$$c_k = \frac{1}{n} \sum_{i=1}^{n-k} (X_i - \bar{X}(n)) (X_{i+k} - \bar{X}(n)) \quad (\text{B.5})$$

Notice that the numerator in Equation (B.4) 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) indicate that for large n , $\text{Var}(r_k) \approx \frac{1}{n}$. 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 $\text{Var}(r_k) \approx \frac{1}{100} = 0.01$. 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.

As we can see from Figure B.5, there does not appear to be any significant correlation with respect to observation number for the computer lab arrival data. The autocorrelation is well-contained within the lag correlation limits denoted with the dashed lines within the plot.

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).

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

ACF Plot for Computer Lab Arrivals

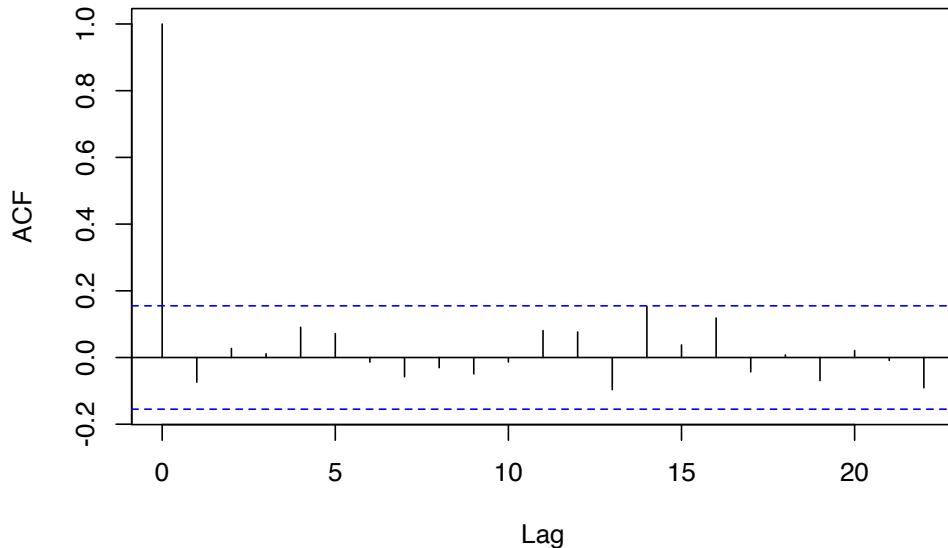


Figure B.5.: Autocorrelation Function Plot for Computer Lab Arrivals

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)
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 B.6 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 multi-factor contingency table test provided by using the *summary* command on the output object, *myTable* of the *xtabs* command.

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 ,

B. Probability Distribution Modeling

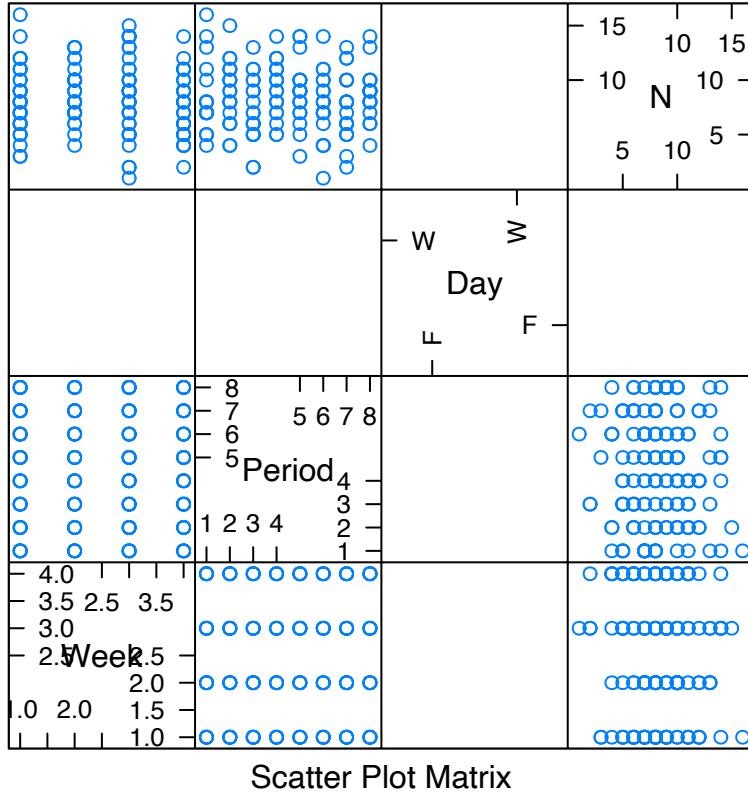


Figure B.6.: Scatter Plot Matrix from Lattice Package for Computer Lab Arrivals

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 p-value is very high, we should not reject H_0 . What does this all mean? In essence, we can now treat the arrival count observation 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 non-homogeneous (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.

B.3.3. Estimating the Rate Parameter for the Poisson Distribution

Testing if the Poisson distribution is a good model for this data can be accomplished using various statistics tests for a Poisson 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 1) the method of moments and 2) 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 λ , where λ represents the expected number of arrivals per 15 minute interval.

$$f(n; \lambda) = P\{N = n\} = \frac{e^{-\lambda} (\lambda)^n}{n!}$$

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 probability probability mass function of the sample is:

$$L(\lambda) = g(n_1, n_2, \dots, n_k; \lambda) = f(n_1; \lambda)f(n_2; \lambda) \dots f(n_k; \lambda) = \prod_{i=1}^k f(n_i; \lambda)$$

The (n_1, n_2, \dots, n_k) are observed (known values) and λ is unknown. The function $L(\lambda)$ is called the likelihood function. To estimate the value of λ by the method of maximum likelihood, we must find the value of λ 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:

B. Probability Distribution Modeling

$$\begin{aligned} L(\lambda) &= \prod_{i=1}^k \frac{e^{-\lambda} (\lambda)^{n_i}}{n_i!} \\ &= \frac{e^{-k\lambda} \lambda^{\sum_{i=1}^k n_i}}{\prod_{i=1}^k n_i!} \end{aligned}$$

It can be shown that maximizing $L(\lambda)$ is the same as maximizing $\ln(L(\lambda))$. This is called the log-likelihood function. Thus,

$$\ln(L(\lambda)) = -k\lambda + \ln(\lambda) \sum_{i=1}^k n_i - \sum_{i=1}^k \ln(n_i!)$$

Differentiating with respect to λ , yields,

$$\frac{d\ln(L(\lambda))}{d\lambda} = -k + \frac{\sum_{i=1}^k n_i}{\lambda}$$

When we set this equal to zero and solve for λ , we get

$$\begin{aligned} 0 &= -k + \frac{\sum_{i=1}^k n_i}{\lambda} \\ \hat{\lambda} &= \frac{\sum_{i=1}^k n_i}{k} \end{aligned} \tag{B.6}$$

If the second derivative, $\frac{d^2 \ln L(\lambda)}{d\lambda^2} < 0$ then a maximum is obtained.

$$\frac{d^2 \ln L(\lambda)}{d\lambda^2} = \frac{-\sum_{i=1}^k n_i}{\lambda^2}$$

because the n_i are positive and λ is positive the second derivative must be negative; therefore the maximum likelihood estimator for the parameter of the Poisson distribution is given by Equation (B.6). Notice that this is the sample average of the interval counts, which can be easily computed using the *mean()* function within R.

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 non-linear 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's complete this example using R to fit and test whether or not the Poisson distribution is a good model for this data.

B.3.4. Chi-Squared Goodness of Fit Test for Poisson Distribution

In essence, a distributional test examines the hypothesis $H_0 : X_i \sim F_0$ versus the alternate hypothesis of $H_a : X_i \not\sim F_0$. That is, the null hypothesis is that data come from distribution function, F_0 and the alternative hypothesis that the data are not distributed according to F_0 .

As a reminder, when using a hypothesis testing framework, we can perform the test in two ways: 1) pre-specifying the Type I error and comparing to a critical value or 2) pre-specifying the Type I error and comparing to a p-value. For example, in the first case, suppose we let our Type I 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 I 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 α level at which the observed test statistic is significant. This smallest α 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, α . 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

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

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

B. Probability Distribution Modeling

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, i.e. your α level.

For a discrete distribution, the most common distributional test is the Chi-Squared goodness of fit test, which is the subject of the next section.

B.3.5. Chi-Squared Goodness of Fit Test

The Chi-Square Test divides the range of the data into, k , intervals (or classes) and tests if the number of observations that fall in each interval (or class) is close the expected number that should fall in the interval (or class) given the hypothesized distribution is the correct model. In the case of discrete data, the intervals are called classes and they are mapped to groupings along the domain of the random variable. For example, in the case of a Poisson distribution, a possible set of $k = 7$ classes could be $\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6 \text{ or more}\}$. As a general rule of thumb, the classes should be chosen so that the expected number of observations in each class is at least 5.

Let c_j be the observed count of the observations contained in the j^{th} class. The Chi-Squared test statistic has the form:

$$\chi_0^2 = \sum_{j=1}^k \frac{(c_j - np_j)^2}{np_j} \quad (\text{B.7})$$

The quantity np_j is the expected number of observations that should fall in the j^{th} class 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 χ_0^2 is greater than $\chi_{\alpha, k-s-1}^2$, where s is the *number of estimated parameters* for the hypothesized distribution. $\chi_{\alpha, k-s-1}^2$ is the upper critical value for a Chi-squared random variable χ^2 such that $P\{\chi^2 > \chi_{\alpha, k-s-1}^2\} = \alpha$. The p-value, $P\{\chi_{k-s-1}^2 > \chi_0^2\}$, for this test can be computed in using the following formula:

$$\text{CHISQ.DIST.RT}(\chi_0^2, k - s - 1)$$

In the statistical package *R*, the formula is:

$$\text{pchisq}(\chi_0^2, k - s - 1, lower.tail = FALSE)$$

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.

Let's perform a Chi-squared goodness of fit test of computer lab data for the Poisson distribution. A good first step in analyzing discrete data is to summarize the observations in a table. We can do that easily with the R `table()` function.

```
# tabulate the counts
tCnts = table(p2$N)
tCnts
```

```
## 
##  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
##  1   3   2   7  13  16  21  25  20  21  11   7   5   6   1   1
```

From this analysis, we see that there were no week-period combinations that had 0 observations, 1 that had 1 count, 3 that had 2 counts, and so forth. Now, we will estimate the rate parameter of the hypothesized Poisson distribution using Equation (B.6) and tabulate the expected number of observations for a proposed set of classes.

```
# get number of observations
n = length(p2$N)
n
```

```
## [1] 160
```

```
# estimate the rate for Poisson from the data
lambda = mean(p2$N)
lambda
```

```
## [1] 8.275
```

```
# setup vector of x's across domain of Poisson
x = 0:15
x
```

```
## [1]  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```

B. Probability Distribution Modeling

```
# compute the probability for Poisson
prob = dpois(x, lambda)
prob

## [1] 0.0002548081 0.0021085367 0.0087240706 0.0240638947 0.0497821822
## [6] 0.0823895116 0.1136288681 0.1343255548 0.1389429957 0.1277503655
## [11] 0.1057134275 0.0795253284 0.0548393410 0.0349073498 0.0206327371
## [16] 0.0113823933
```

```
# view the expected counts for these probabilities
prob*n
```

```
## [1] 0.04076929 0.33736587 1.39585130 3.85022316 7.96514916 13.18232186
## [7] 18.18061890 21.49208877 22.23087932 20.44005848 16.91414839 12.72405254
## [13] 8.77429457 5.58517596 3.30123794 1.82118293
```

We computed the probability associated with the Poisson distribution with $\hat{\lambda} = 8.275$. Recall that the Possion distribution has the following form:

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

Then, by multiplying each probability, $P\{X = i\} = p_i$ by the number of observations $n = 160$, we can get the expected number, $n \times p_i$, that we should observed under the hypothesized distribution. Since the expected values for $x = 0, 1, 2, 3, 4$ are so small, we consolidate them in to one class to have the expected number to be at least 5. We will also consolidate the range $x \geq 13$ into a single class and the values of 11 and 12 into a class.

```
# compute the probability for the classes
# the vector is indexed starting at 1, prob[1] = P(X=0)
cProb = c(sum(prob[1:5]), prob[6:11], sum(prob[12:13]), ppois(12, lambda, lower.tail = FALSE))
cProb

## [1] 0.08493349 0.08238951 0.11362887 0.13432555 0.13894300 0.12775037 0.10571343
## [8] 0.13436467 0.07795112
```

```
# compute the expected counts for each of the classes
expected = cProb*n
expected

## [1] 13.58936 13.18232 18.18062 21.49209 22.23088 20.44006 16.91415 21.49835
## [9] 12.47218
```

Thus, we will use the following $k = 9$ classes: {0,1,2,3,4}, {5}, ..., {10}, {11,12}, and {13 or more}. Now, we need to summarize the observed frequencies for the proposed classes.

```
# transform tabulated counts to data frame
dfCnts = as.data.frame(tCnts)
# extract only frequencies
cnts = dfCnts$Freq
cnts

## [1] 1 3 2 7 13 16 21 25 20 21 11 7 5 6 1 1
```

```
# consolidate classes for observed
observed = c(sum(cnts[1:4]), cnts[5:10], sum(cnts[11:12]), sum(cnts[13:16]))
observed

## [1] 13 13 16 21 25 20 21 18 13
```

Now, we have both the observed and expected tabulated and can proceed with the chi-squared goodness of fit test. We will compute the result directly using Equation (B.7).

```
# compute the observed minus expected components
chisq = ((observed - expected)^2)/expected
# compute the chi-squared test statistic
sumchisq = sum(chisq)
# chi-squared test statistic
print(sumchisq)

## [1] 2.233903
```

B. Probability Distribution Modeling

```
# set the degrees of freedom, with 1 estimated parameter s = 1
df = length(expected) - 1 - 1
# compute the p-value
pvalue = 1 - pchisq(sumchisq, df)
# p-value
print(pvalue)
```

```
## [1] 0.9457686
```

Based on such a high p-value, we would not reject H_0 . Thus, we can conclude that there is not enough evidence to suggest that the lab arrival count data is some other distribution than the Poisson distribution. In this section, we did the Chi-squared test *by-hand*. R has a package that facilitates distribution fitting called, *fitdistrplus*. We will use this package to analyze the computer lab arrival data again in the next section.

B.3.6. Using the *fitdistrplus* R Package on Discrete Data

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 probability mass function and the cumulative distribution function as illustrated in Figure B.7.

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.

```
fp = fitdist(p2$N, "pois")
summary(fp)
plot(fp)
gofstat(fp)
```

```
fp = fitdist(p2$N, "pois")
summary(fp)
```

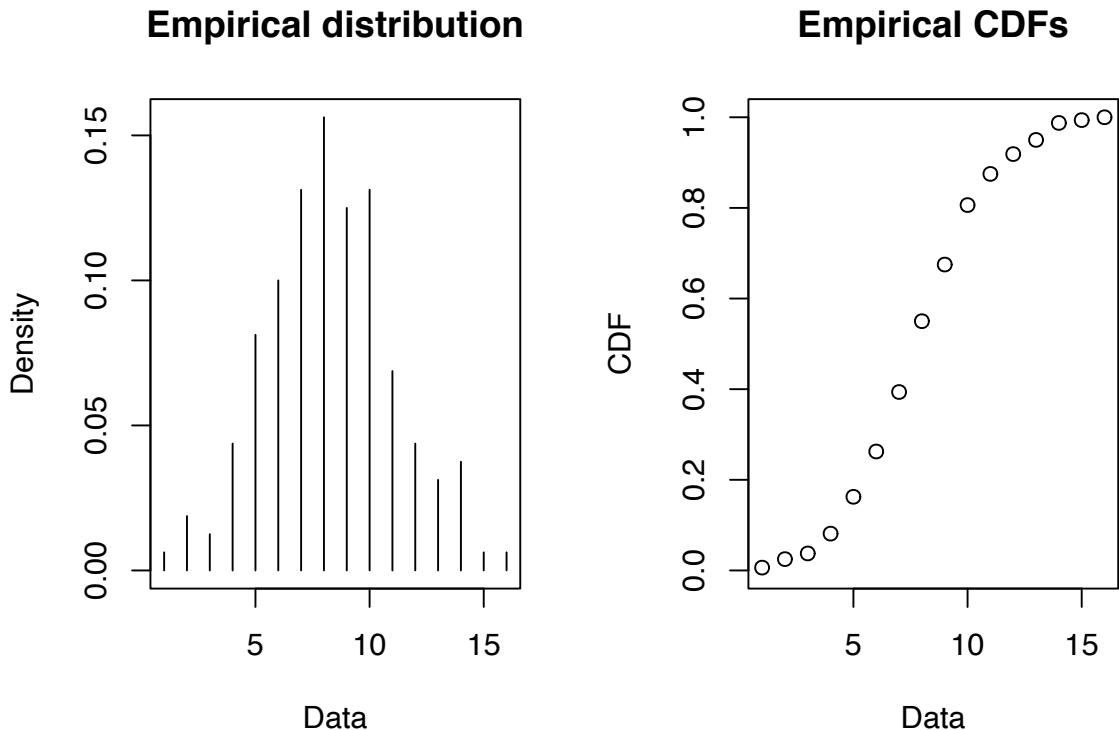


Figure B.7.: Plot of the Empirical PMF and CDF of the Computer Lab Arrivals

```
## 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
```

```
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
```

B. Probability Distribution Modeling

```

## <= 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
## Akaike's Information Criterion 789.5485
## Bayesian Information Criterion 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*.

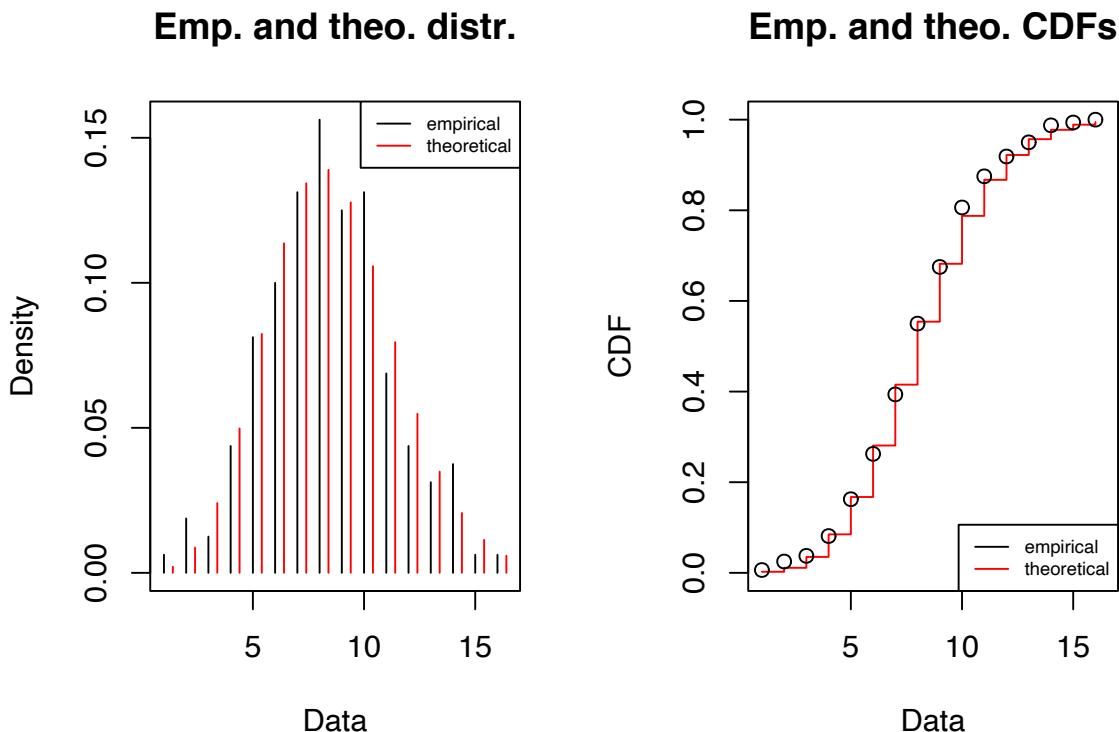


Figure B.8.: Plot of the Empirical and Theoretical CDF of the Computer Lab Arrivals

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). These are exactly the same results that we computed in the previous section. 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 lab arrival counts.

B.3.7. 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 as shown in Table B.4

Table B.4.: Loading Dock 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

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.

```
packageCnt = scan("data/AppDistFitting/TruckLoadingData.txt")
hist(packageCnt, main="Packages per Shipment", xlab="#packages")
stripchart(packageCnt, method="stack", pch="o")
table(packageCnt)
```

As we can see from the *R* output, the range of the data varies between 100 and 150. The histogram, shown in Figure B.9, illustrates the shape of the data. It appears to be slightly positively skewed. Figure B.10 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.

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 show in the *R* listing.

```
tp = table(packageCnt)
tp/length(packageCnt)
```

```
## packageCnt
##   100   110   120   130   140   150
## 0.050 0.150 0.225 0.375 0.125 0.075
```

B. Probability Distribution Modeling

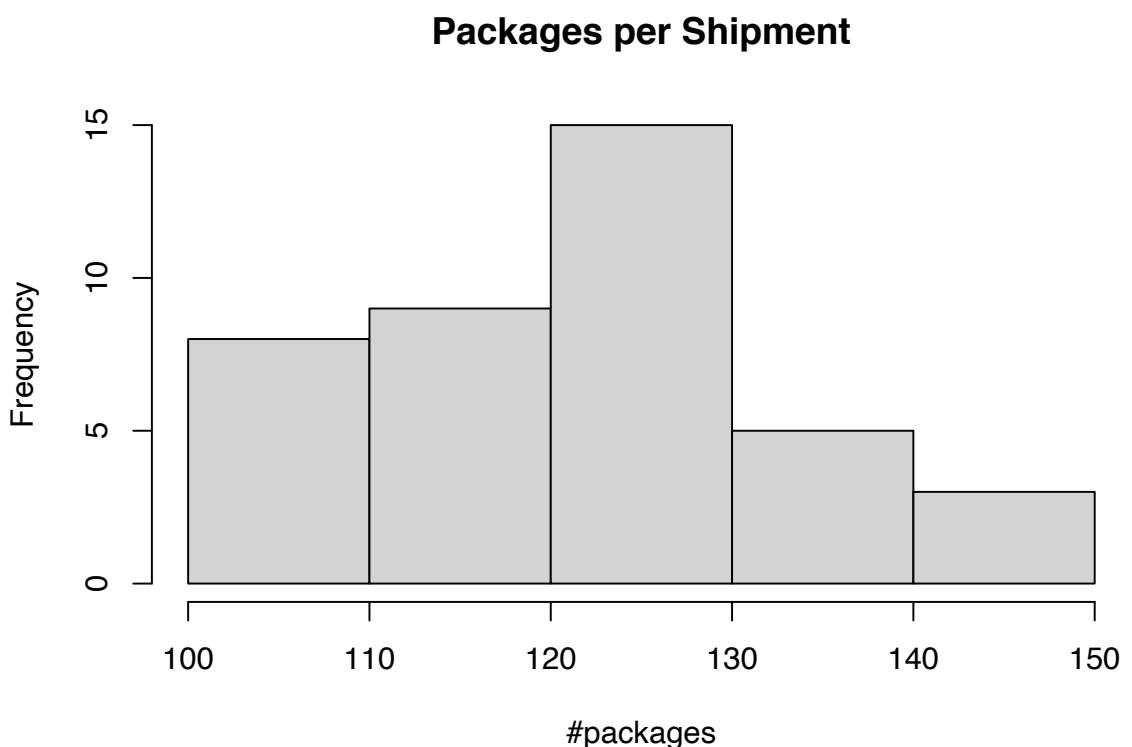


Figure B.9.: Histogram of Package Count per Shipment

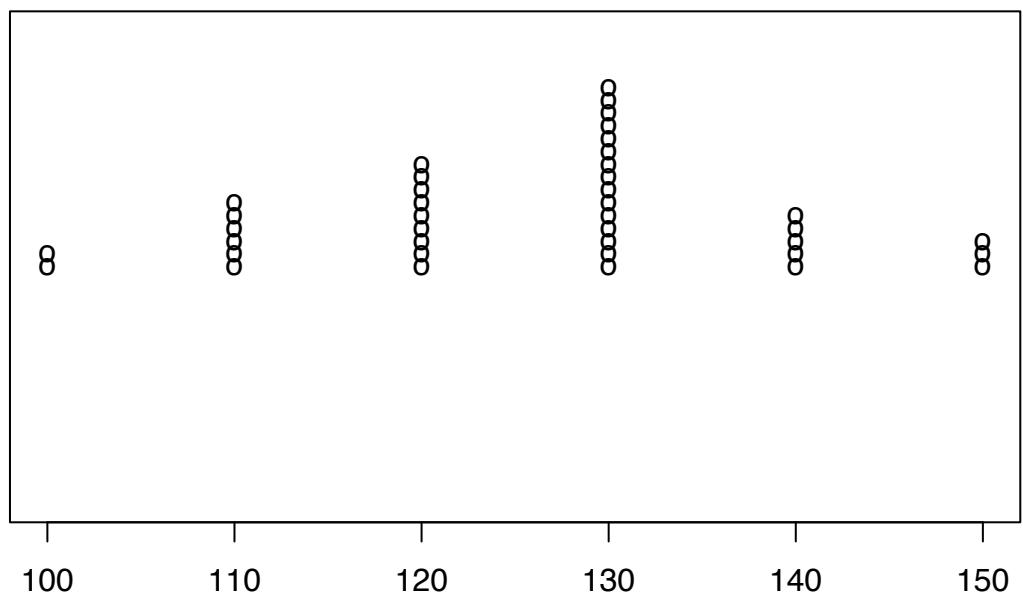


Figure B.10.: Dot Plot of Package Counter Shipment

Thus, we can represent this situation with the following probability mass and cumulative distribution functions.

$$P\{X = x\} = \begin{cases} 0.05 & x = 100 \\ 0.15 & x = 110 \\ 0.225 & x = 120 \\ 0.375 & x = 130 \\ 0.125 & x = 140 \\ 0.075 & x = 150 \end{cases}$$

$$F(x) = \begin{cases} 0.0 & \text{if } x < 100 \\ 0.05 & \text{if } 100 \leq x < 110 \\ 0.20 & \text{if } 110 \leq x < 120 \\ 0.425 & \text{if } 120 \leq x < 130 \\ 0.80 & \text{if } 130 \leq x < 140 \\ 0.925 & \text{if } 140 \leq x < 150 \\ 1.0 & \text{if } x \geq 150 \end{cases}$$

The previous two examples illustrated the process for fitting a discrete distribution to data for use in a simulation model. Because discrete event dynamic simulation involves modeling the behavior of a system over time, the modeling of distributions that represent the time to perform a task is important. Since the domain of time is on the set of positive real numbers, it is a continuous variable. We will explore the modeling of continuous distributions in the next section.

B.4. Modeling with Continuous Distributions

Continuous distributions 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 F.2 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. Alternatively, if you have no a priori knowledge that some events are more likely than others, then a uniform distribution seems like a reasonable starting point. 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:

B. Probability Distribution Modeling

$$E[X] = \frac{a + b}{2}$$

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

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 ± 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, fifty percent 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).

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 λ has:

$$E[X] = \frac{1}{\lambda}$$

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

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 Δt 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 Δt . 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? Two reasons: 1) it often is a good model for many situations found in nature and 2) 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 A.2.4. 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 B.5 lists common modeling situations for various continuous distributions.

Table B.5.: 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

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. In the following sections, we will illustrate how to fit continuous distributions to

B. Probability Distribution Modeling

data.

B.5. Fitting Continuous Distributions

Previously, we examined the modeling of discrete distributions. In this section, we will 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 B.2 (Fitting a Gamma Distribution). Suppose that we are interested in modeling the time that it takes to perform a computer component repair task. The 100 observations are provide 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

B.5.1. 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, *task-Times.txt* within the R working directory.

```

y = scan(file="data/AppDistFitting/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")

```

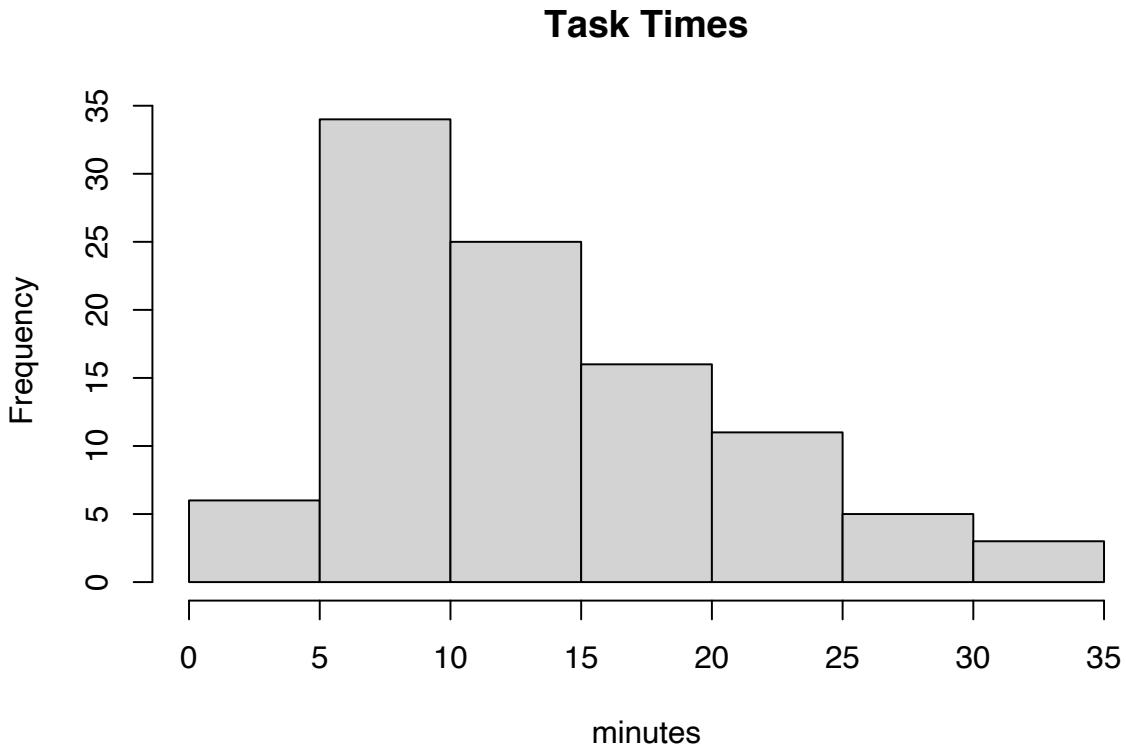


Figure B.11.: Histogram of Computer Repair Task Times

As can be seen in Figure B.11, the histogram is slightly right skewed.

The time series plot, Figure B.12, illustrates no significant pattern (e.g. trends, etc.).

Finally, the autocorrelation plot, Figure B.13, shows no significant correlation (the early lags are well within the confidence band) with respect to observation number.

Based on the visual analysis, we can conclude that the task times are likely to be independent and identically distributed.

B.5.2. 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. The *summary* command summarizes the distributional properties in terms of the minimum, maximum, median, and 1st and 3rd quartiles of the data.

B. Probability Distribution Modeling

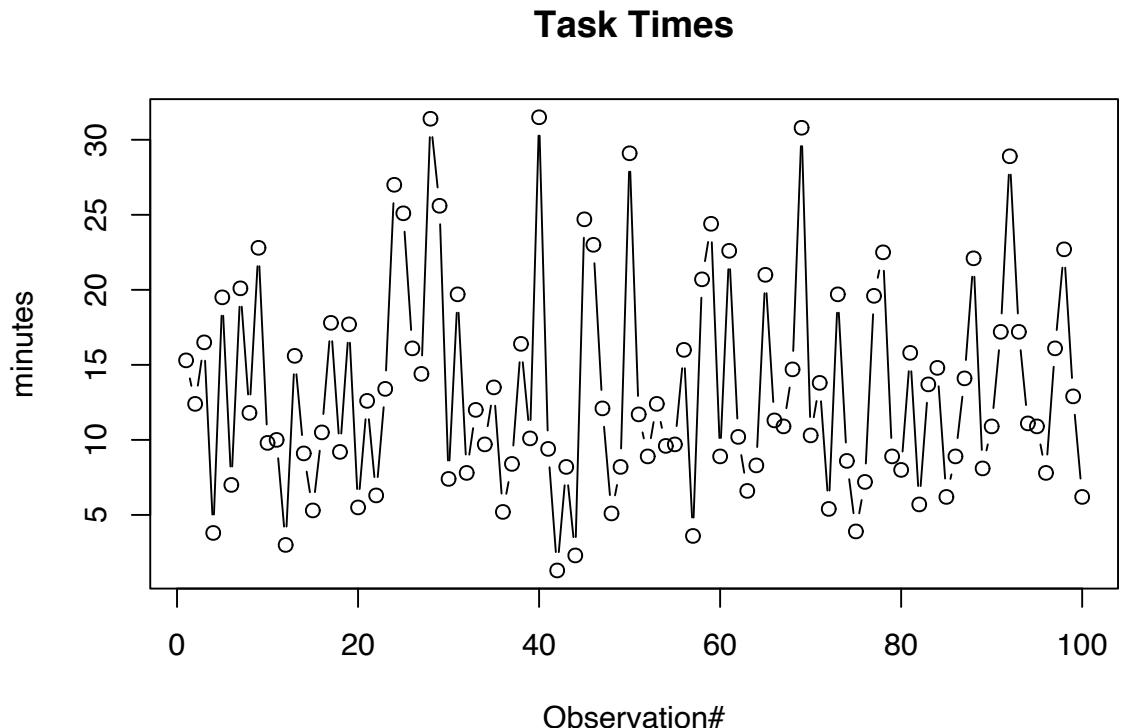


Figure B.12.: Time Series Plot of Computer Repair Task Times

```
summary(y)
```

```
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## 1.300   8.275  11.750  13.412  17.325  31.500
```

The *mean*, *var*, *sd* commands compute the sample average, sample variance, and sample standard deviation of the data.

```
mean(y)
```

```
## [1] 13.412
```

```
var(y)
```

```
## [1] 50.44895
```

ACF Plot for Task Times

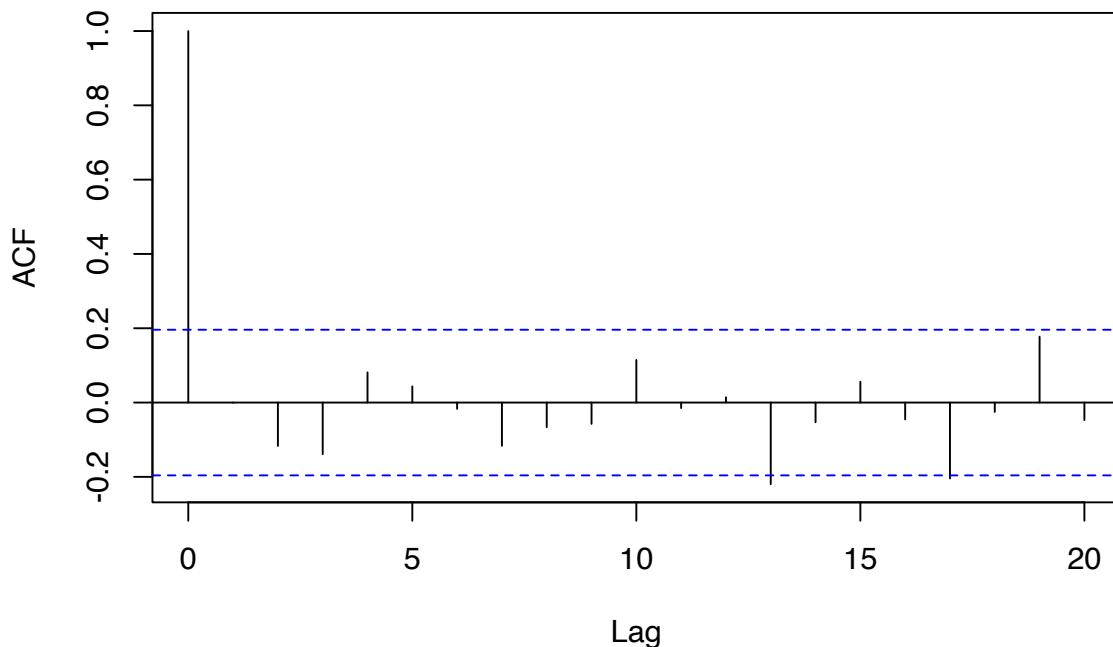


Figure B.13.: ACF Plot of Computer Repair Task Times

```
sd(y)
```

```
## [1] 7.102742
```

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.

```
t.test(y)
```

```
##
## One Sample t-test
##
## data: y
## t = 18.883, df = 99, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
```

B. Probability Distribution Modeling

```
## 12.00266 14.82134
## sample estimates:
## mean of x
## 13.412
```

The *descdist* command of the *fitdistrplus* package will also provide a description of the distribution's properties.

```
descdist(y, graph=FALSE)
```

```
## 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's 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.

B.5.3. 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.

Here 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 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, define the following function:

$$c(\vec{x} \leq b) = \#\{x_i \leq b\} \quad i = 1, \dots, n \quad (\text{B.8})$$

$c(\vec{x} \leq b)$ 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:

$$c_j = c(\vec{x} \leq b_j) - c(\vec{x} \leq b_{j-1}) \quad (\text{B.9})$$

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.

$$p_j = P\{b_{j-1} \leq X \leq b_j\} = \int_{b_{j-1}}^{b_j} f(x)dx = \Delta b \times f(y) \text{ for } y \in (b_{j-1}, b_j)$$

Therefore, since h_j is an estimate for p_j , the shape of the distribution should be proportional to the relative frequency, i.e. $h_j \approx \Delta b \times f(y)$.

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 width of the intervals are too large, the resulting histogram will be very block like. Two common rules for setting the number of interval are:

B. Probability Distribution Modeling

1. Square root rule, choose the number of intervals, $k = \sqrt{n}$.
2. Sturges rule, choose the number of intervals, $k = \lfloor 1 + \log_2(n) \rfloor$.

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.

```
# make histogram with no plot
h = hist(y, plot = FALSE)
# show the histogram object components
h

## $breaks
## [1] 0 5 10 15 20 25 30 35
##
## $counts
## [1] 6 34 25 16 11 5 3
##
## $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"
```

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.

```
h$counts
```

```
## [1] 6 34 25 16 11 5 3
```

The breakpoints are given by the variable `h$breaks`. Note that by default `hist` defines the intervals as right-closed, i.e. $(b_{k-1}, b_k]$, 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, i.e. hcounts/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 (for example: `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.

```
# set up some new break points
b = c(0,4,8,12,16,20,24,28,32)
b
```

```
## [1] 0 4 8 12 16 20 24 28 32
```

```
# make histogram with no plot for new breakpoints
hb = hist(y, breaks = b, plot = FALSE)
# show the histogram object components
hb
```

```
## $breaks
## [1] 0 4 8 12 16 20 24 28 32
##
## $counts
## [1] 6 16 30 17 12 9 5 5
##
## $density
## [1] 0.0150 0.0400 0.0750 0.0425 0.0300 0.0225 0.0125 0.0125
##
## $mids
```

B. Probability Distribution Modeling

```
## [1] 2 6 10 14 18 22 26 30
##
## $xname
## [1] "y"
##
## $equidist
## [1] TRUE
##
## attr(),"class")
## [1] "histogram"
```

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.

```
#define the intervals
y.cut = cut(y, breaks=b)
# tabulate the counts in 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
```

By 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:

$$\chi_0^2 = \sum_{j=1}^k \frac{(c_j - np_j)^2}{np_j} \quad (\text{B.10})$$

where

$$p_j = P\{b_{j-1} \leq X \leq b_j\} = \int_{b_{j-1}}^{b_j} f(x)dx = F(b_j) - F(b_{j-1}) \quad (\text{B.11})$$

Notice that p_j depends on $F(x)$, the cumulative distribution function 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 (α) and the scale (β) 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, sample variance, etc.) 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:

$$E[X] = \alpha\beta$$

$$Var[X] = \alpha\beta^2$$

Setting $\bar{X} = E[X]$ and $S^2 = Var[X]$ and solving for α and β yields,

$$\hat{\alpha} = \frac{(\bar{X})^2}{S^2}$$

$$\hat{\beta} = \frac{S^2}{\bar{X}}$$

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 B.7 illustrates the computations necessary to compute the chi-squared test statistic.

Table B.7.: 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	∞	0.00	0.99	1.00	0.01	1.03	1.03

B. Probability Distribution Modeling

Since the 7th and 8th intervals have less than 5 expected counts, we should combine them with the 6th interval. Computing the chi-square test statistic value over the 6 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.

```
a = mean(y)*mean(y)/var(y) #estimate alpha
b = var(y)/mean(y) #estmate beta
hy = hist(y, plot=FALSE) # make histogram
LL = hy$breaks # set lower limit of intervals
UL = c(LL[-1],10000) # set upper limit of intervals
FLL = pgamma(LL,shape = a, scale = b) #compute F(LL)
FUL = pgamma(UL,shape = a, scale = b) #compute F(UL)
pj = FUL - FLL # compute prob of being in interval
ej = length(y)*pj # compute expected number in interval
e = c(ej[1:5],sum(ej[6:8])) #combine last 3 intervals
cnts = c(hy$counts[1:5],sum(hy$counts[6:7])) #combine last 3 intervals
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
```

```
## [1] 2.742749
```

```
print(pvalue) #print p-value
```

```
## [1] 0.4330114
```

Notice that we computed the same χ^2 value and p-value as when doing the calculations by hand.

B.5.4. Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov (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 cumulative distribution function (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) indicates 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 (B.8), we can define the empirical distribution as in Equation (B.12).

$$\tilde{F}_n(x) = \frac{c(\vec{x} \leq x)}{n} \quad (\text{B.12})$$

To formalize this definition, suppose we have a sample of data, x_i for $i = 1, 2, \dots, n$ and we 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 (B.12) can be re-written as:

$$\tilde{F}_n(x_{(i)}) = \frac{i}{n} \quad (\text{B.13})$$

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:

B. Probability Distribution Modeling

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

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 in the statistical package R.

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

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

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. Table G.5 contains critical values for the K-S test, where you reject the null hypothesis if D_n is greater than the critical value D_α , where α is the Type I significance level.

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, i.e. 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.

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 = 0.052654, 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, i.e. 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) describes 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.

B. Probability Distribution Modeling

B.5.5. 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 by x_q and is defined by:

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

As shown in Figure B.14, 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 cumulative distribution function.

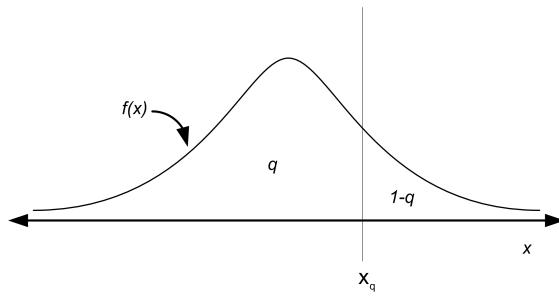


Figure B.14.: The Quantile of a Distribution

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. Thus, the quantile can be defined as:

$$x_q = F^{-1}(q)$$

where F^{-1} represents the inverse of the cumulative distribution function (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 cumulative distribution function, you get back the corresponding ordinate value that is associated with the area under the curve, e.g. 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 Q-Q plots for this situation.

B. Probability Distribution Modeling

```

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 B.15 illustrates the P-P plot.

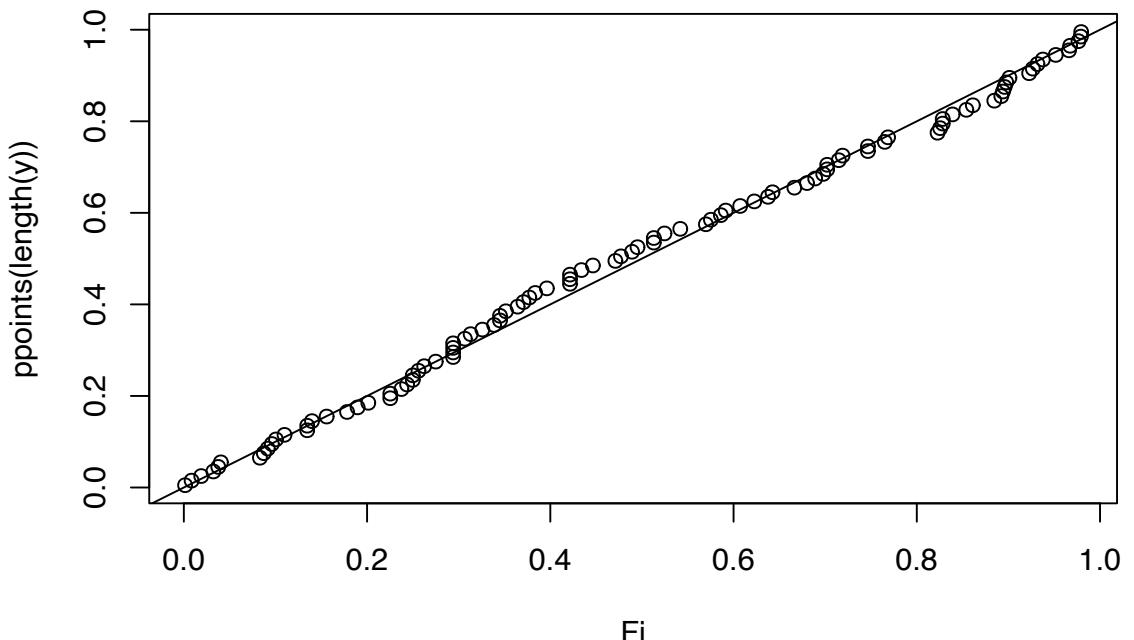


Figure B.15.: The P-P Plot for the Task Times with $\text{gamma}(\text{shape} = 3.56, \text{scale} = 3.76)$

The Q-Q plot should appear approximately linear with intercept zero and slope 1, i.e. a 45 degree 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 B.16 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’s see how easy it can be using the `fitdistrplus` package. Notice that the `fitdist` command will fit the parameters of the distribution.

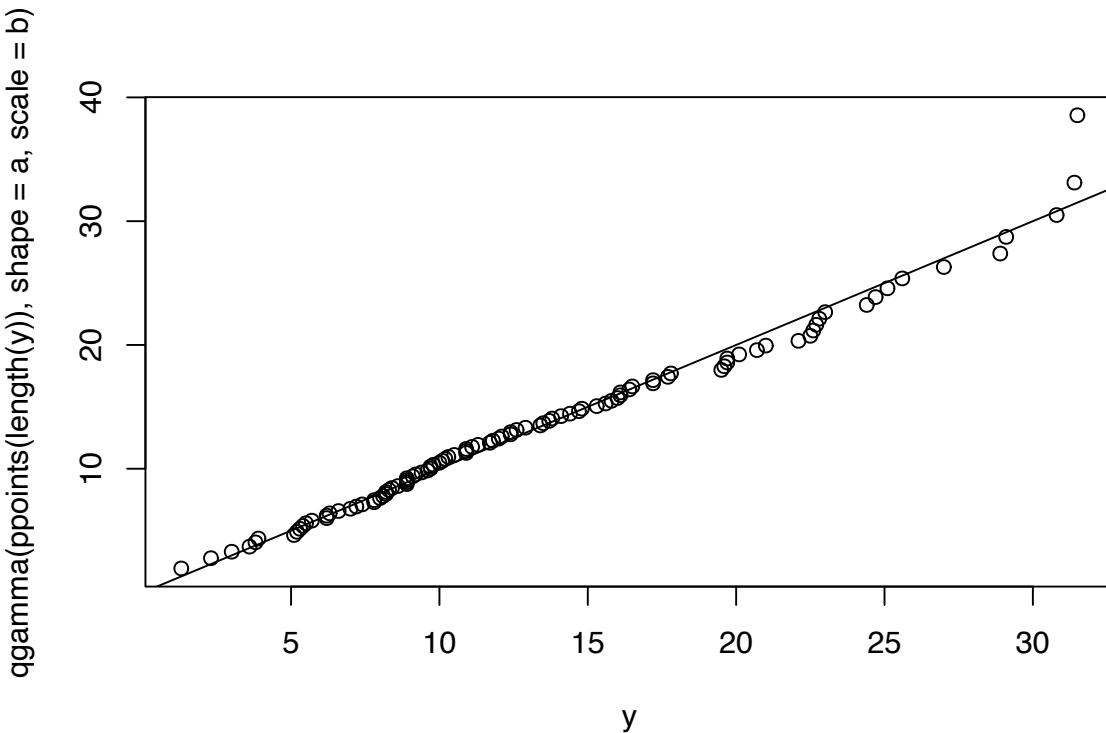


Figure B.16.: The Q-Q Plot for the Task Times with $\text{gamma}(\text{shape} = 3.56, \text{scale} = 3.76)$

```
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
```

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.

```
gfy = gofstat(fy)
print(gfy)
```

B. Probability Distribution Modeling

```
## 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
## Akaike's Information Criterion 663.3157
## Bayesian Information Criterion 668.5260
```

```
print(gfy$chisq) # chi-squared test statistic
```

```
## [1] 3.544766
```

```
print(gfy$chisqvalue) # chi-squared p-value
```

```
## [1] 0.8956877
```

```
print(gfy$chisqdf) # chi-squared degrees of freedom
```

```
## [1] 8
```

Plotting the object returned from the *fitdist* command via (e.g. *plot(fy)*), produces a plot (Figure B.17) of the empirical and theoretical distributions, as well as the P-P and Q-Q plots.

Figure B.18 illustrates the P-P and Q-Q plots if we were to hypothesize a uniform distribution. Clearly, the plots in Figure B.18 illustrate that a uniform distribution is not a good model for the task times.

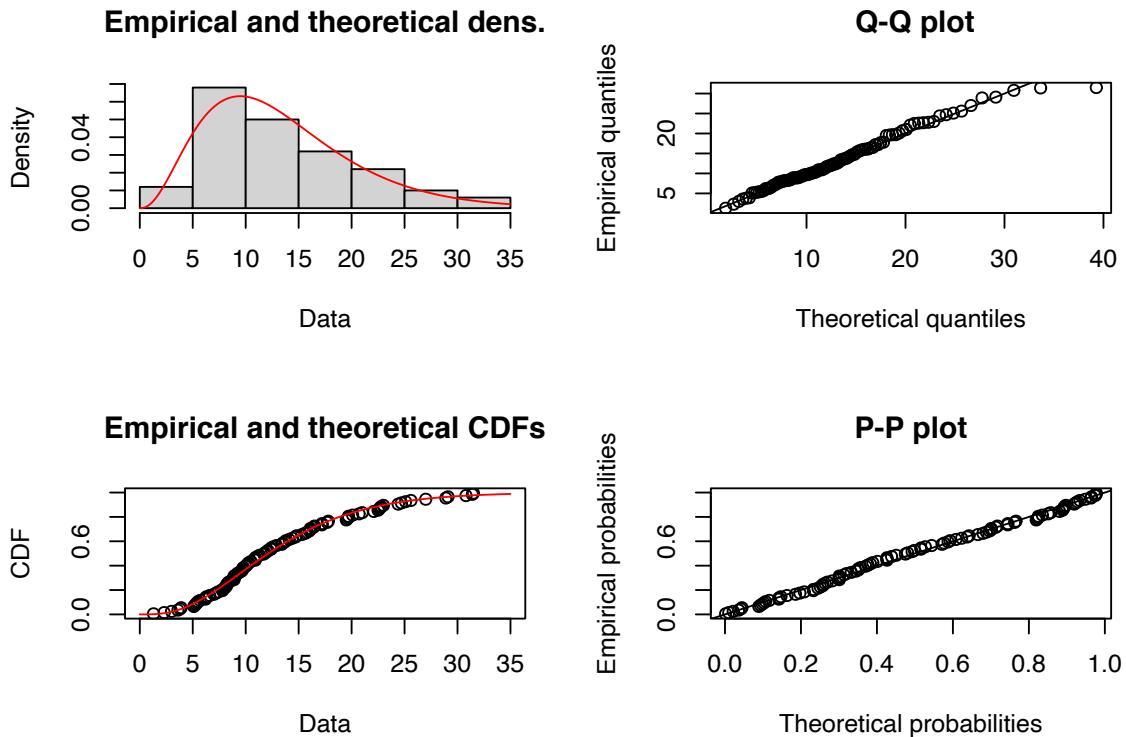


Figure B.17.: Distribution Plot from `fitdistrplus` for Gamma Distribution Fit of Computer Repair Times

B.5.6. Using the Input Analyzer

In this section, we will use the Arena Input Analyzer to fit a distribution to service times collected for the pharmacy example. The Arena Input Analyzer is a separate program that comes with Arena. It is available as part of the free student edition of Arena.

Let X_i be the service time of the i^{th} customer, where the service time is defined as starting when the $(i - 1)^{st}$ 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 B.8 where the first observation is in row 1 column 1, the second obser-

B. Probability Distribution Modeling

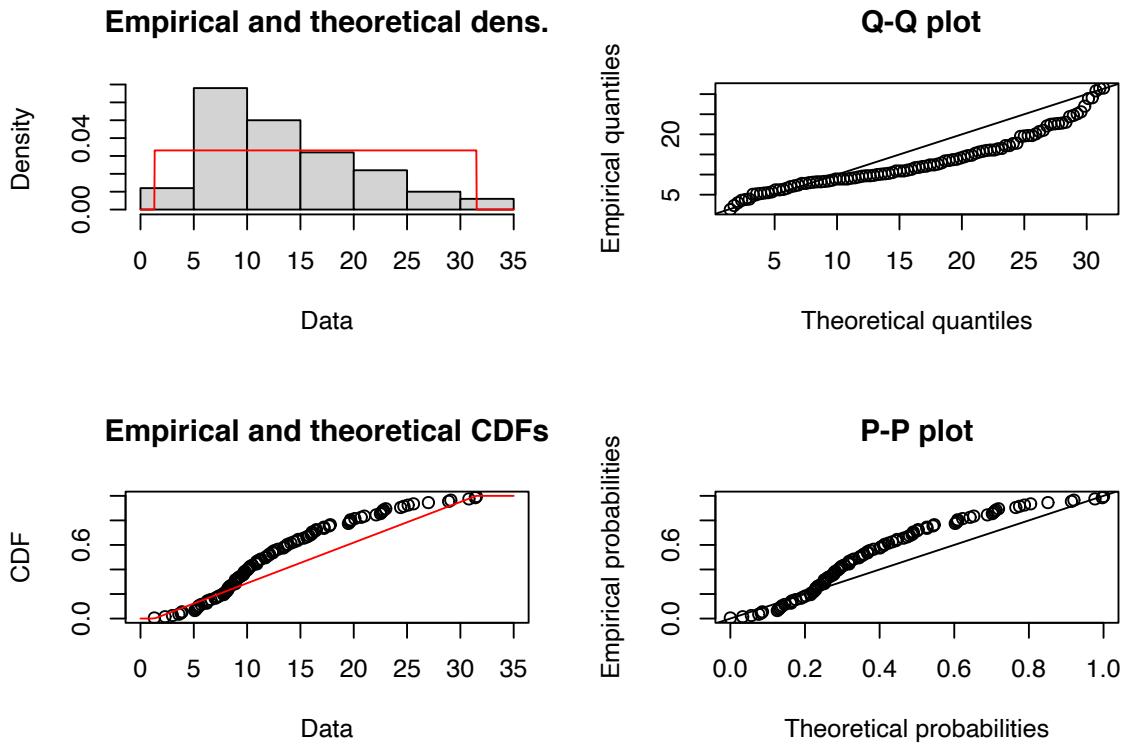


Figure B.18.: Distribution Plot from *fitdistrplus* for Uniform Distribution Fit of Computer Repair Times

vation 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.

Table B.8.: 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

Table B.8.: Pharmacy Service Times

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

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 using R in the previous sections.

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 like Figure B.19.

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 B.19, 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 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 1) individually for a specific distribution and 2) by fitting all of the possible distributions. Given the interval specification the Input Analyzer will compute a Chi-Squared goodness of fit statistic, Kolmogorov-Smirnov Test, and squared error criteria, all of which will be discussed in what follows.

Let's 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 the following listing.

```
Distribution Summary
Distribution: Exponential
Expression: 36 + EXP0(147)
Square Error: 0.003955
```

```
Chi Square Test
```

B. Probability Distribution Modeling

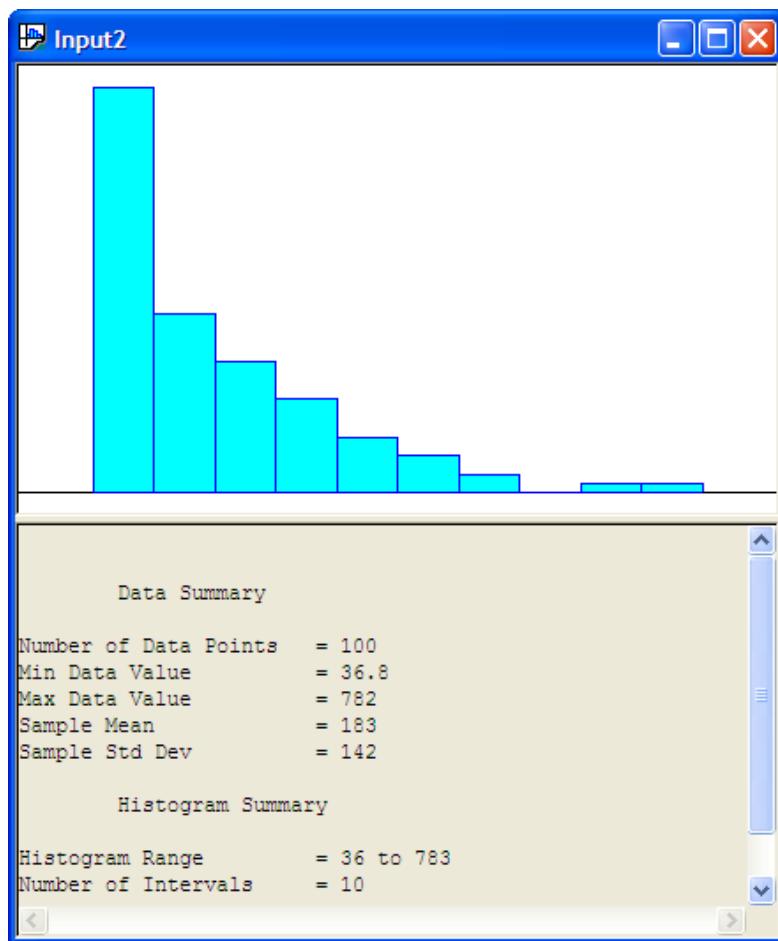


Figure B.19.: Input Analyzer After Data Import

```
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
```

The Input Analyzer has made a fit to the data and has recommended the Arena expression (36 + EXPO(147)). What is this value 36? The value 36 is called the offset or location parameter. The visual fit of the data is shown in Figure B.20

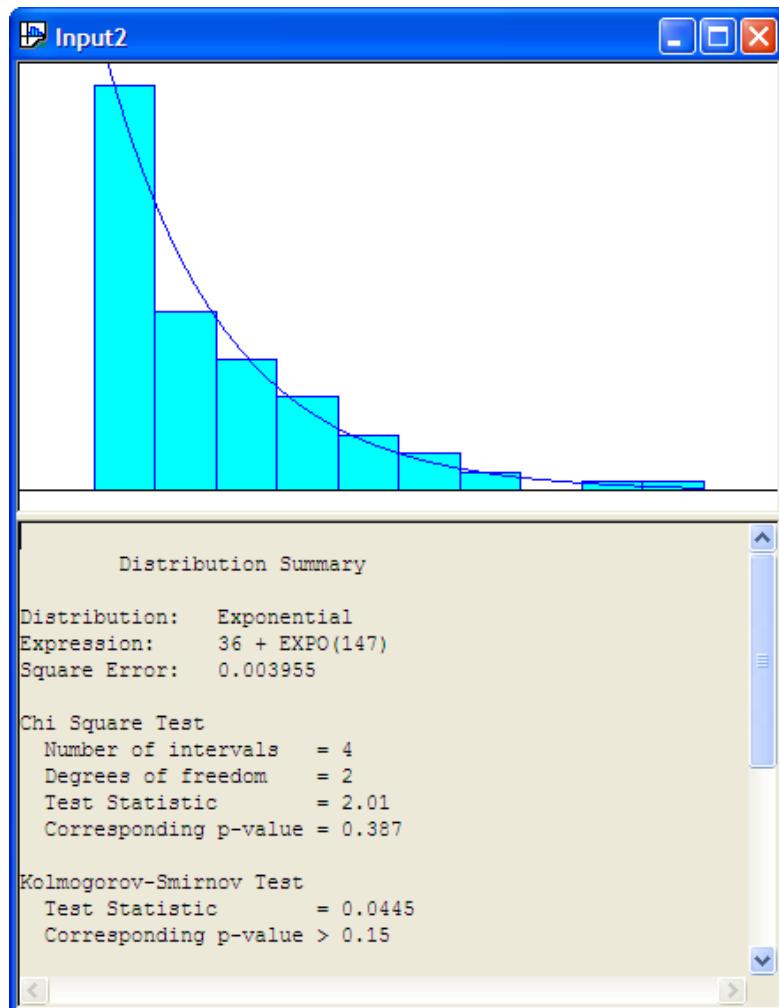


Figure B.20.: Histogram for Exponential Fit to Service Times

Recall the discussion in Section A.2.4 concerning shifted distributions. Any distribution can have this additional parameter that shifts it along the x-axis. This can complicate parameter estimation procedures. The Input Analyzer has an algorithm which will attempt to estimate this

B. Probability Distribution Modeling

parameter. Generally, a reasonable estimate of this parameter can be computed via the floor of the minimum observed value, $\lfloor \min(x_i) \rfloor$. 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 Kolmogorov-Smirnov Test. The null hypothesis is that the data come from they hypothesized distribution versus the alternative hypothesis that the data do not come from the hypothesized distribution. The Input Analyzer shows the p-value of the tests.

The results of the distribution fitting process indicate 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 propose 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 B.21 show the results of fitting a uniform distribution to the data.

The following listing shows the results for the uniform distribution. The results show 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.

```
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
```

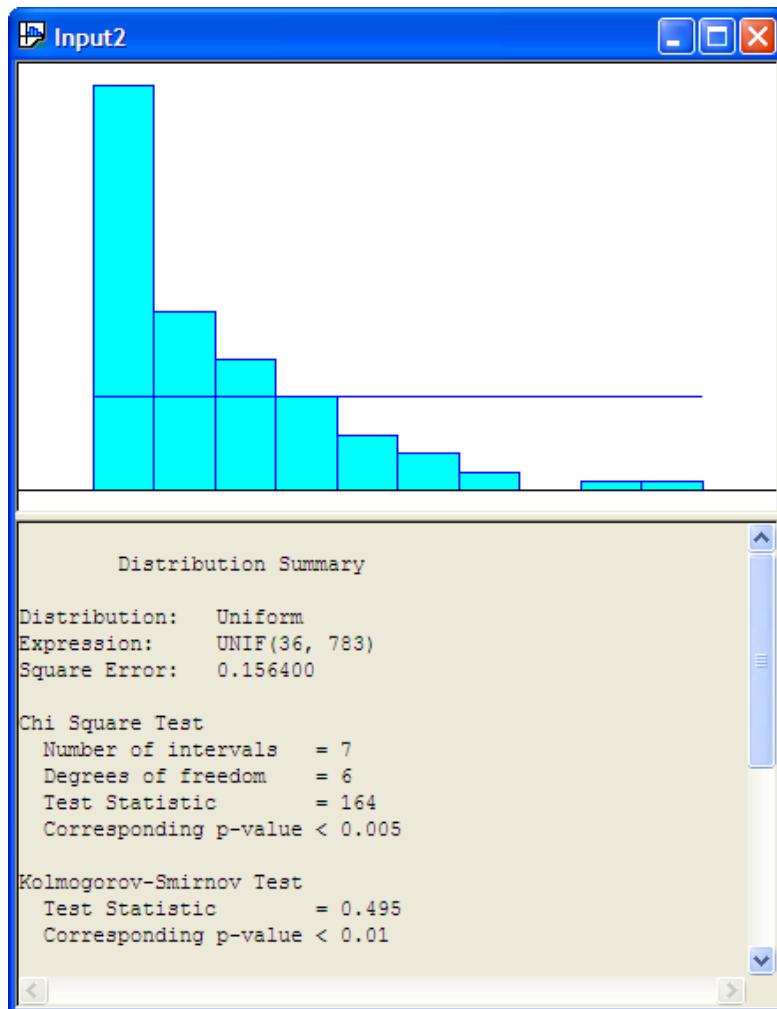


Figure B.21.: Uniform Distribution and Histogram for Service Time Data

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

B. Probability Distribution Modeling

defined as the sum over the intervals of the squared difference between the relative frequency and the probability associated with each interval:

$$\text{Square Error} = \sum_{j=1}^k (h_j - \hat{p}_j)^2 \quad (\text{B.14})$$

Table B.9 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.

Table B.9.: 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

When you select 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 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 B.22). The Window > Fit All Summary menu option will show the squared error criteria for all the distributions that were fit. Figure B.23 indicates that the Erlang distribution is second in the fitting process according to the squared error criteria and the Exponential distribution is third in the rankings. Since the Exponential distribution is a special case of the Erlang distribution we see that their squared error criteria is the same. Thus, in reality, these results reflect the same distribution.

By using Options > Parameters > Histogram, the Histogram Parameters dialog can be used to change the parameters associated with the histogram as shown in Figure B.24.

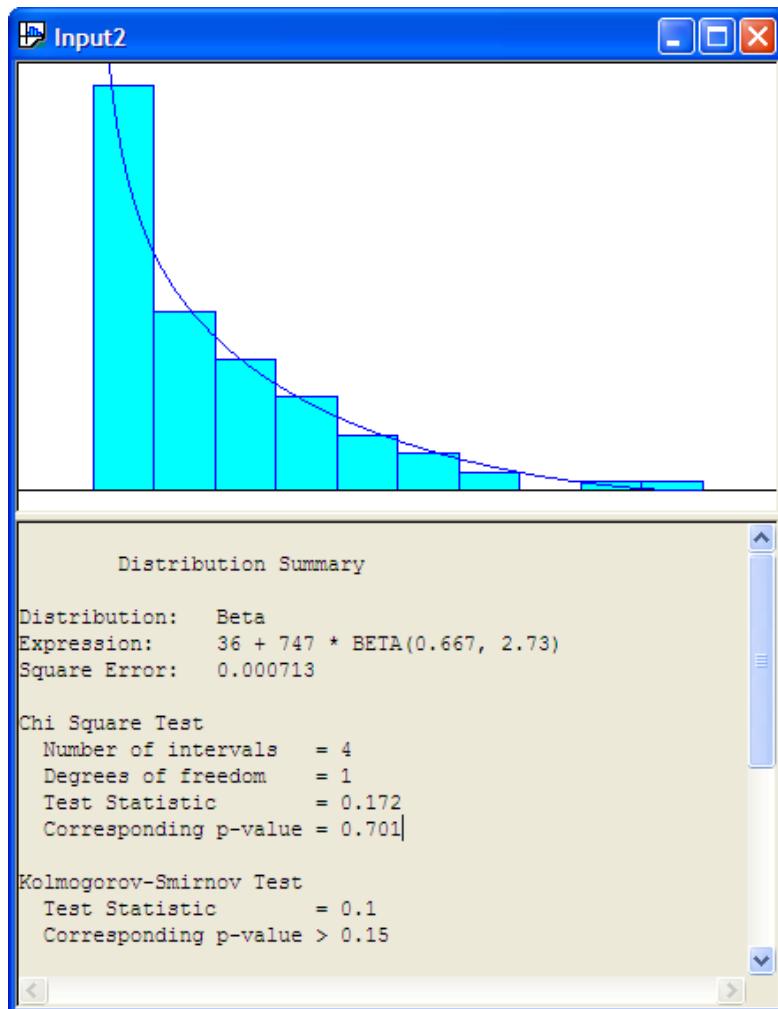


Figure B.22.: Fit All Beta Recommendation for Service Time Data

Changing the number of intervals to 12 results in the output provided in Figure B.25, which indicates that the exponential distribution is a reasonable model based on the 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. 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 vari-

B. Probability Distribution Modeling

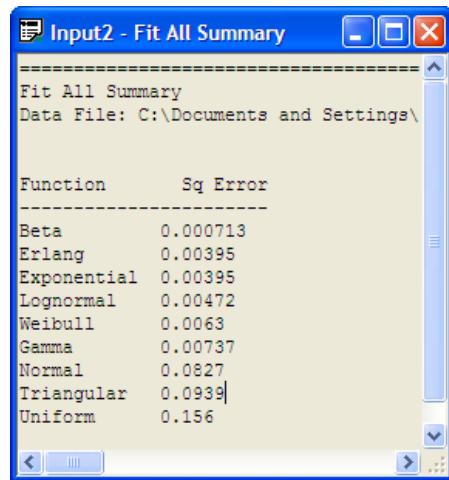


Figure B.23.: Fit All Recommendation for Service Time Data

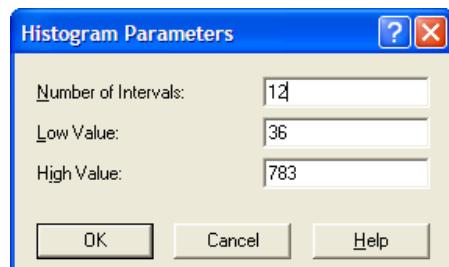


Figure B.24.: Changing the Histogram Parameters

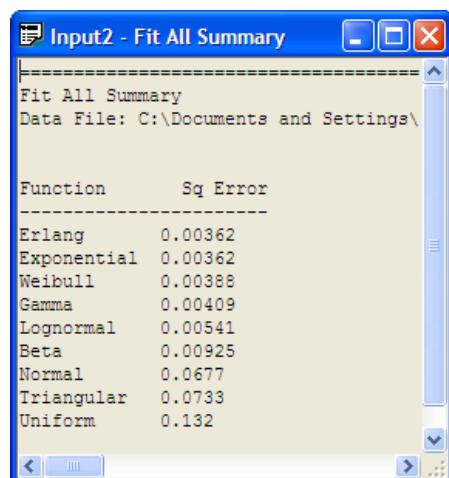


Figure B.25.: Fit All with 12 Intervals

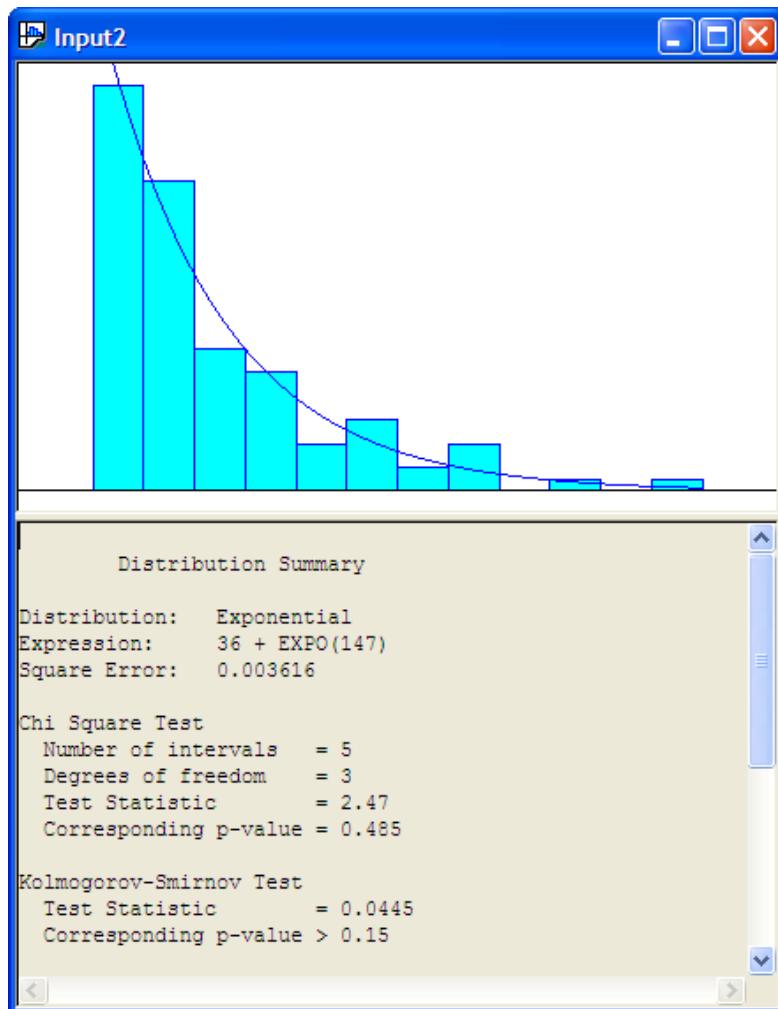


Figure B.26.: Exponential Fit with 12 Intervals

able that represents the service time then $X \sim 36 + \text{EXPO}(147)$, where 147 is 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 B.27 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.

```
x = scan(file="ServiceTimes.txt") #read in the file
Read 100 items
```

B. Probability Distribution Modeling

```
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)
```

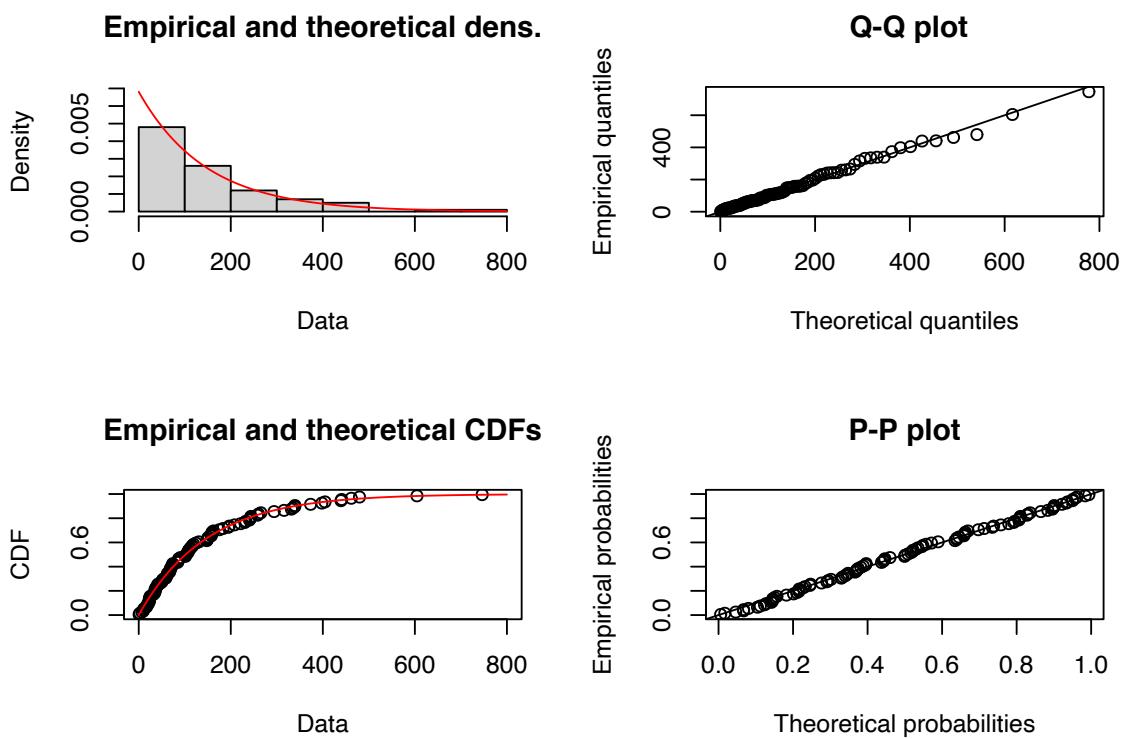


Figure B.27.: Distribution Plot from fitdistrplus for Service Time Data

The P-P and Q-Q plots of the shifted data indicate that the exponential distribution is an excellent fit for the service time data.

B.6. Testing Uniform (0,1) Pseudo-Random Numbers

Now that we have seen the general process for fitting continuous distributions, this section will discuss the special case of testing for uniform (0,1) random variables. The reason that this is important is because these methods serve the basis for testing if pseudo-random numbers can reasonably be expected to perform as if they are U(0,1) random variates. Thus, this section provides 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), that are utilized. Typical tests examine:

- Distributional properties: e.g. Chi-Squared and Kolmogorov-Smirnov test
- Independence: e.g. Correlation tests, runs tests
- Patterns: e.g. Poker test, 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. Just like for general continuous distributions, the two major tests that are utilized to examine the distributional properties of sequences of pseudo-random numbers are the Chi-Squared Goodness of Fit test and the Kolmogorov-Smirnov test. In the case of testing for $U(0, 1)$ random variates some simplifications can be made in computing the test statistics.

B.6.1. Chi-Squared Goodness of Fit Tests for Pseudo-Random Numbers

When applying the Chi-Squared goodness of fit to test 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 α

B. Probability Distribution Modeling

If p_j values are chosen as $\frac{1}{k}$, then Equation (B.15) can be rewritten as:

$$\chi_0^2 = \frac{k}{n} \sum_{j=1}^k \left(c_j - \frac{n}{k} \right)^2 \quad (\text{B.15})$$

Let's apply these concepts to a small example.

Example B.3 (Testing 100 Pseudo-Random Numbers). Suppose we have 100 observations from a pseudo-random number generator. Perform a χ^2 test that the numbers are distributed $U(0, 1)$.

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 $n = 100$ observations, the number of intervals should be less than or equal to 20. Let's choose $k = 10$. This means that $p_j = 0.1$ for all j . The following table summarizes the computations for each interval for computing the chi-squared test statistic.

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

B.6. Testing Uniform (0,1) Pseudo-Random Numbers

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	$\frac{(c_j - np_j)^2}{np_j}$
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 $U(0, 1)$. This process can be readily implemented within a spreadsheet or performed using R. Assuming that the file, u01data.txt, contains the PRNs for this example, then the following R commands will perform the test:

```
data = scan(file="data/AppDistFitting/u01data.txt") # read in the file
b = seq(0,1, by = 0.1) # set up the break points
h = hist(data, b, right = FALSE, plot = FALSE) # tabulate the counts
chisq.test(h$counts) # perform the test
```

```
##
## Chi-squared test for given probabilities
##
## data: h$counts
## X-squared = 8.2, df = 9, p-value = 0.5141
```

Because we are assuming that the data is $U(0, 1)$, the chisq.test() function within R is simplified because its default is to assume that all data is equally likely. Notice that we get exactly the same result as we computed when doing the calculations manually.

B.6.2. Higher Dimensional Chi-Squared Test

The pseudo-random numbers should not only be uniformly distributed on the interval $(0, 1)$ 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 .

B. Probability Distribution Modeling

The serial test, described in (Law, 2007) can be used to assess the quality of the higher dimensional properties of pseudo-random numbers. Suppose the sequence of pseudo-random numbers can be formed into non-overlapping vectors each of size d . The vectors should be independent and identically distributed random vectors uniformly distributed on the d -dimensional unit hyper-cube. This motivates the development of the serial test for uniformity in higher dimensions. To perform the serial test:

1. Divide $(0,1)$ into k sub-intervals of equal size.
2. Generate n vectors of pseudo-random 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

$$\chi_0^2(d) = \frac{k^d}{n} \sum_{j_1=1}^k \sum_{j_2=1}^k \dots \sum_{j_d=1}^k \left(c_{j_1, j_2, \dots, j_d} - \frac{n}{k^d} \right)^2 \quad (\text{B.16})$$

5. Reject the hypothesis that the \vec{U}_i 's are uniformly distributed in the d -dimensional unit hyper-cube if $\chi_0^2(d) > \chi_{\alpha, k^d-1}^2$ or if the p-value $P\{\chi_{k^d-1}^2 > \chi_0^2(d)\}$ is less than α .

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

Example B.4 (2-D Chi-Squared Test in R for $U(0,1)$). Using the same data as in the previous example, perform a 2-Dimensional χ^2 Test using the statistical package R. Use 4 intervals for each of the dimensions.

The following code listing is liberally commented for understanding the commands and essentially utilizes some of the classification and tabulation functionality in R to compute Equation (B.16). The code displayed here is available in the files associated with the chapter in file, 2dchisq.R.

```

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")

```

The results shown in the following listing 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 2-D intervals are close to the expected. The p-value suggests that we would not reject the hypothesis that there is non-uniformity in the pairs of the psuedo-random numbers. 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.

B. Probability Distribution Modeling

```

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

```

B.6.3. Kolmogorov-Smirnov Test for Pseudo-Random Numbers

When applying the K-S test to testing pseudo-random 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:

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

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:

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

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

Example B.5 (K-S Test for U(0,1)). Given the data from Example B.3 test the hypothesis that the data appears $U(0, 1)$ versus that it is not $U(0, 1)$ using the Kolmogorov-Smirnov 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 third 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
\times	\times	\times	\times	\times	\times	\times
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

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

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

Thus, we have that

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

B. Probability Distribution Modeling

Referring to Table G.5 and using the approximation for sample sizes greater than 35, we have that $D_{0.05} \approx 1.36/\sqrt{n}$. Thus, $D_{0.05} \approx 1.36/\sqrt{100} = 0.136$. Since $D_n < D_{0.05}$, we would not reject the hypothesis that the data is uniformly distribution over the range from 0 to 1.

The K-S test performed in the solution to Example B.5 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 = scan(file="data/AppDistFitting/u01data.txt") # read in the file
ks.test(data,"punif",0,1) # perform the test
```

```
##  
## One-sample Kolmogorov-Smirnov test  
##  
## data: data  
## D = 0.10809, 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 $U(0, 1)$.

B.6.4. Testing for Independence and Patterns in Pseudo-Random Numbers

A full treatment for testing for independence and patterns within sequences of pseudo-random numbers is beyond the scope of this text. However, we will illustrate some of the basic concepts in this section.

As previously noted an analysis of the autocorrelation structure associated with the sequence of pseudo-random numbers forms a basis for testing dependence. A sample autocorrelation plot can be easily developed once the autocorrelation values have been estimated. 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* for the data in Example B.3. Using the `*acf()$` function in *R* makes it easy to get estimates of the autocorrelation estimates. The resulting plot is shown in Figure B.28. 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.

Autocorrelation Plot of Pseudo-Random Number Data

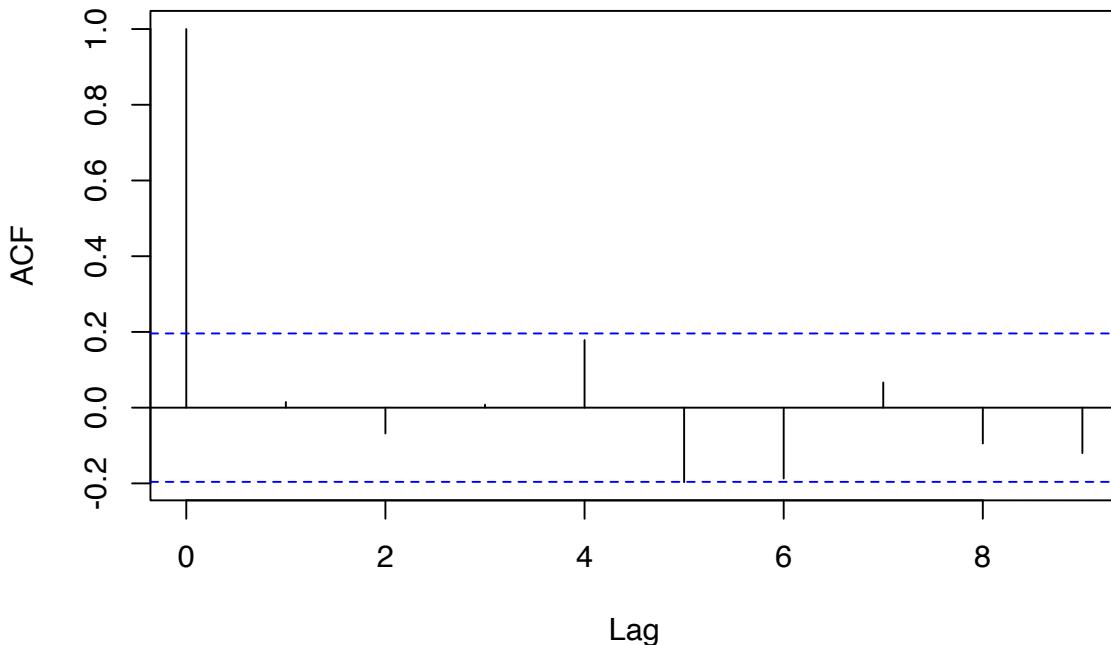


Figure B.28.: Autocorrelation Plot for u01data.txt Data Set

```
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
```

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, 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. The Table B.13 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 a 8 runs (4 runs up and 4 runs down).

B. Probability Distribution Modeling

Table B.13.: 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
-	+	+	+	-	-	-	-	+	-	+	+	+	-	+

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, three of a kind, etc.). Banks et al. (2005) discusses how to perform these tests.

Does all this testing really matter? Yes! You should always know what pseudo-random number generator you are using within your simulation models and you should always know if the generator has passed a battery of statistical tests. The development and testing of random number generators is serious business. You should stick with well researched generators and reliable well tested software.

B.7. Additional Distribution 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 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 can't 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 B.15. Again, this limits us to only the values observed in the sample.

Table B.15.: 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

One can also use the continuous empirical distribution, which interpolates between the distribution values.

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

B. Probability Distribution Modeling

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 queueing 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 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. An introduction to the topic is provided in (Banks et al., 2005).

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 non-homogeneous 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 1.8 presents a distribution for the service times based on this method.

Table B.16.: 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, ∞)	0.2	1.0

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.

B.8. 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:

- Understanding how the system being modeled works. This understanding improves overall model construction in an iterative fashion: 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 in to your simulations

Properly modeling the inputs to the simulation model form a critical foundation to increasing the validity of the simulation model and subsequently, the credibility of the simulation outputs.

B.9. Exercises

The files referenced in the exercises are available in the files associated with this chapter.

Exercise B.1. The observations available in the text file, *problem1.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.

Exercise B.2. The observations available in the text file, *problem2.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.

Exercise B.3. The observations available in the text file, *problem3.txt*, represent the time in minutes that it takes a repair person 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.

Exercise B.4. The observations available in the text file, *problem4.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.

Exercise B.5. The observations available in the text file, *problem5.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.

Exercise B.6. The observations available in the text file, *problem6.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.

Exercise B.7. If $Z \sim N(0, 1)$, and $Y = \sum_{i=1}^k Z_i^2$ then $Y \sim \chi_k^2$, where χ_k^2 is a chi-squared random variable with k degrees of freedom. Setup a model to generate $n = 32, 64, 128, 256, 1024$ observations of Y with $k = 5$. For each sample, fit a distribution to the sample.

Exercise B.8. Consider the following sample (also found in *problem8.txt*) that represents the time (in seconds) that it takes a hematology cell counter 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

- a. Test the hypothesis that these data are drawn from a uniform distribution at a 95% confidence level assuming that the interval is between 0 and 100.
- b. The interval of the distribution is between a and b , where a and b are unknown parameters estimated from the data.
- c. 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

B. Probability Distribution Modeling

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

Exercise B.9. Consider the following frequency data on the number of orders received per day by a warehouse.

j	c_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		
	Totals	100		

- Compute the sample mean for this data.
 - Perform a χ^2 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.
-

Exercise B.10. 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 and also found in file *problem10.txt*:

14	16	14	16	12	14	12	13	12	12
12	12	14	12	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.

Exercise B.11. Test the following fact by generating instances of $Y = \sum_{i=1}^r X_i$, where $X_i \sim \text{expo}(\beta)$ and $\beta = E[X_i]$. Using $r = 5$ and $\beta = 2$. Be careful to specify the correct parameters for the exponential distribution function. The exponential distribution takes in the mean of the distribution as its parameter. Generate 10, 100, 1000 instances of Y . Perform hypothesis tests to check $H_0 : \mu = \mu_0$ versus $H_1 : \mu \neq \mu_0$ where μ is the true mean of Y for each of the sample sizes generated. Use the same samples to fit a distribution using the Input Analyzer. Properly interpret the statistical results supplied by the Input Analyzer.

Exercise B.12. 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 and also found in file *problem12.csv*

Obs#	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

B. Probability Distribution Modeling

1. Visualize the data.
 2. Check if the day of the week influences 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 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.
-

Exercise B.13. 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 data file *problem13.txt*

C. Queueing Theory

LEARNING OBJECTIVES

- To be able understand basic queueing theory notation
- To be able to compute queueing results for single queue situations
- To be able to identify and apply standard queueing models

Many real-life situations involve the possible waiting of entities (e.g. customers, parts, etc.) for resources (e.g. bank tellers, machines, etc.). Systems that involve waiting lines are called queueing systems. This appendix introduces analytical (formula-based) approaches to modeling 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 situation of modeling a drive through pharmacy, 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 these situations, having more of the resource (pharmacist) 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 queueing 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. 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. The purpose of this appendix is not to provide a comprehensive treatment of queueing theory for which there are many excellent books already. The purpose of this appendix is to provide an introduction to this topic for persons new to simulation so that they can better understand their modeling and analysis of queueing systems. This appendix also serves as a reference for the formulas associated with these models to facilitate their use in verifying and validating simulation models.

C.1. Single Line Queueing Stations

Chapter 2 presented the pharmacy model and analyzed it with a single server, single queue queueing system called the M/M/1. This section shows how the formulas for the M/M/1 model

C. Queueing Theory

in Chapter 2 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 queueing system, there are customers that 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. Queueing 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 queueing theory involve a single line that is served by a set of servers. Figure C.1 illustrates the major components of a queueing system with a single queue feeding into a set of servers.

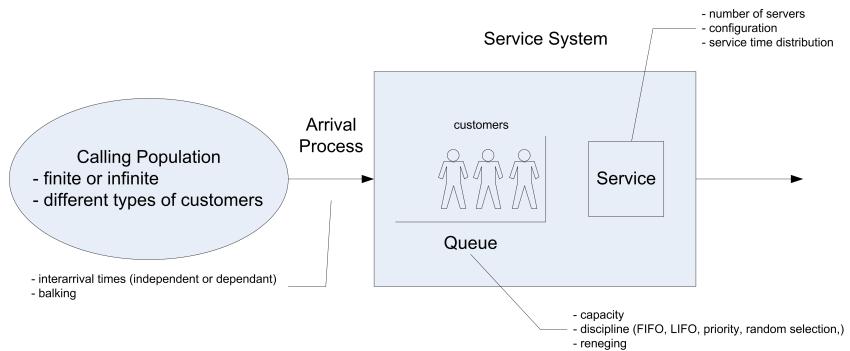


Figure C.1.: Example single queue system

In queueing 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 C.1, the potential customers can be described as coming from a calling population. The term calling population comes from the historical use of queueing 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 upon 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 inter-arrival times and in particular the inter-arrival time distribution. In general, the calling population may also have different types of customers that 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 last-in, first out 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 that are receiving service. Each customer that arrives to the system may require a particular number of units of the resource. The resource component of this system can have 1 or more servers. In the analytical treatment, each customer will required 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. Queueing systems that contain servers that operate in this fashion are often referred to as parallel server systems. When a customer arrives to the system, the customer will either be placed in the queue or placed 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.

C.1.1. Queueing 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 that there is 1 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 that arrives. The

C. Queueing Theory

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 2 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, i.e. that the time between arrivals are independent and identically distributed and that the service times are also independent and identically distributed.

Also, the standard models assume that the arrival process is independent of the service process and vice a 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 5 servers. Figure C.2 illustrates some of the common queueing systems as denoted by Kendall's notation.

There are a number of quantities that form the basis for measuring the performance of queueing systems:

C.1. Single Line Queueing Stations

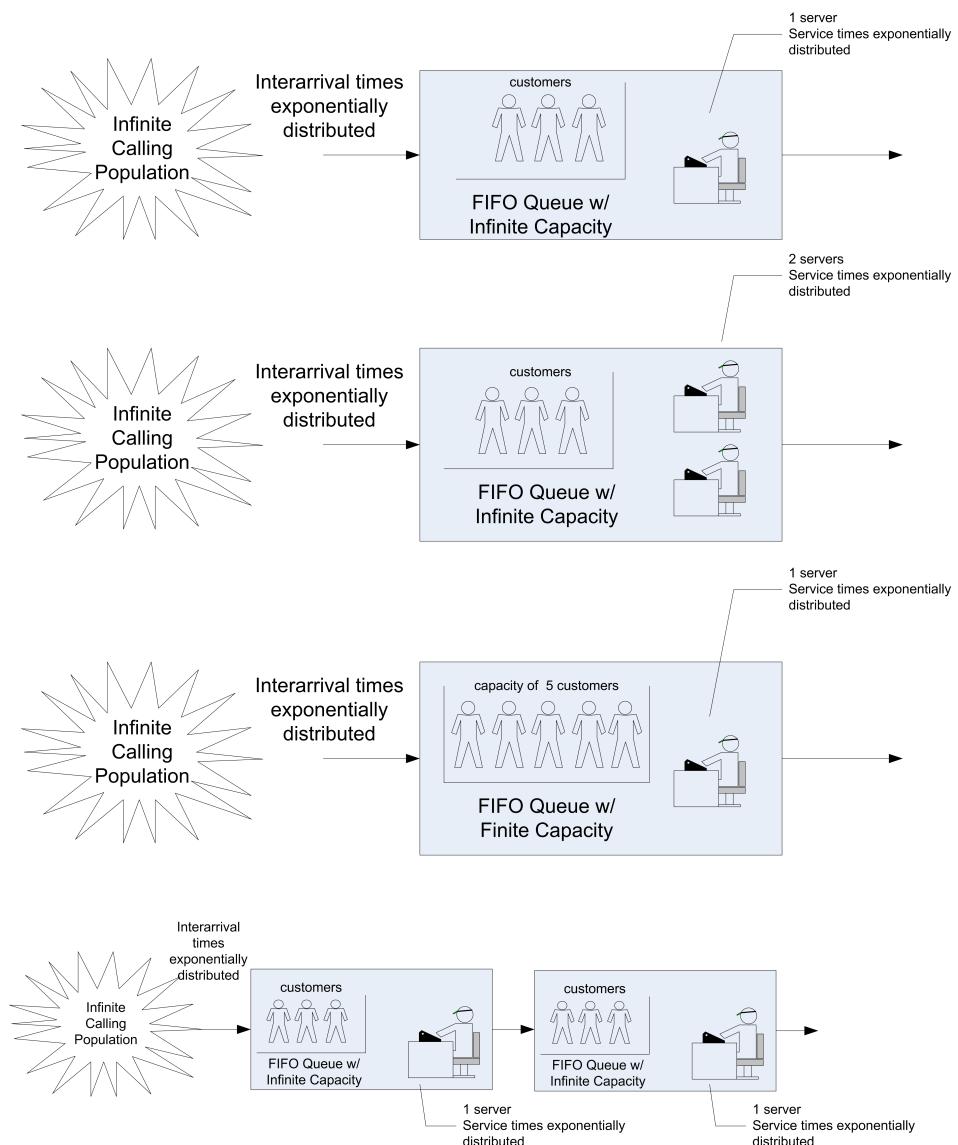


Figure C.2.: Illustrating some common queueing situations

C. Queueing Theory

- 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 queueing system has stochastic elements. We will assume that the system is work conserving, i.e. that the customers do not exit without received all of their required service and that a customer uses one and only one server to receive service. For a work conserving queue, the following is true:

$$N(t) = N_q(t) + N_b(t)$$

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 2 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:

$$E[N(t)] = E[N_q(t)] + E[N_b(t)]$$

If ST is the service time of an arbitrary customer, then it should be clear that:

$$E[T] = E[T_q] + E[ST]$$

That is, the expected system time is equal to the expected waiting time in the queue plus the expected time spent in service.

C.1. Single Line Queueing Stations

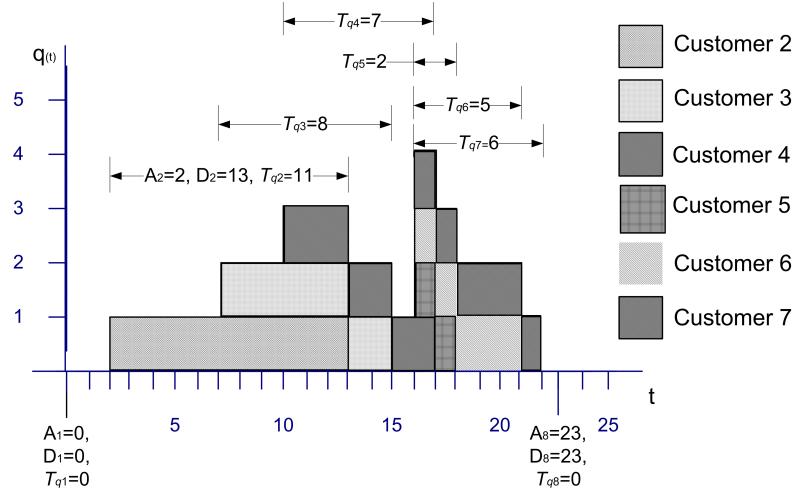


Figure C.3.: Sample path for the number of customers in a queue

C.1.2. Little's Formula

Chapter 3 presented time persistent data and computed time-averages. For queueing 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. To understand these relationships, consider Figure C.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:

$$\bar{T}_q = \frac{\sum_{i=1}^n T_{q_i}}{n} = \frac{0 + 11 + 8 + 7 + 2 + 5 + 6 + 0}{8} = \frac{39}{8} = 4.875$$

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:

C. Queueing Theory

$$\sum_{i=1}^n T_{q_i} = 39$$

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 that exit the queue over a period of time. That is,

$$\bar{R} = \frac{n}{t_n - t_0}$$

where n is the number of customers that 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:

$$\bar{L}_q = \frac{\int_{t_0}^{t_n} q(t)dt}{t_n - t_0} = \frac{n}{t_n - t_0} \times \frac{\sum_{i=1}^n T_{q_i}}{n} = \bar{R} \times \bar{T}_q$$

This relationship is a conservation law and can also be applied to other portions of the queueing 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 queueing 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 queueing theorist who first formalized the technical conditions of its applicability to the stochastic processes within queues of this nature. The interested reader is

C.1. Single Line Queueing Stations

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 , 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 λ be the expected arrival rate so that $1/\lambda$ is the mean of the inter-arrival 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:

$$\begin{aligned} L &\equiv E[N] \\ L_q &\equiv E[N_q] \\ B &\equiv E[N_b] \\ W &\equiv E[T] \\ W_q &\equiv E[T_q] \end{aligned}$$

Thus, it should be clear that

$$\begin{aligned} L &= L_q + B \\ W &= W_q + E[ST] \end{aligned}$$

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:

$$\begin{aligned} L &= \lambda W \\ L_q &= \lambda W_q \\ B &= \lambda E[ST] = \frac{\lambda}{\mu} \end{aligned}$$

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 that 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 λ 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 that you would see (on average) when looking behind you. Thus, $L_q = \lambda W_q$.

Notice that λ and μ 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 1 server while in service, B also represents the expected number of busy servers in steady state. If there

C. Queueing Theory

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:

$$\rho = \frac{B}{c} = \frac{\lambda}{c\mu}$$

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 queueing 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.

C.1.3. 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) are 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's 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 that 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 that are in the phone booth at any time t . Clearly, the possible values of $N(t)$ are 0 and 1, i.e. $N(t) \in \{0, 1\}$. Developing formulas for the probability that there are 0 or 1 customers in the booth at any time t , i.e. $P_i(t) = P\{N(t) = i\}$ will be the key to modeling this situation.

Let λ 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 $\lambda = 5/\text{hr}$, and if the mean length of a call is 10 minutes, then $\mu = 6/\text{hr}$. The following reasonable assumptions will be made:

1. The probability of a customer arriving in a small interval of time, Δt , is roughly proportional to the length of the interval, with the proportionality constant equal to the mean rate of arrival, λ .
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, μ .
3. The probability of more than one arrival in an arbitrarily small interval, Δt , is negligible. In other words, Δt , 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, Δt , is negligible. In other words, Δt , 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 Δt , or there could have been one customer in the system at time t and the customer completed service during Δt . 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 Δt , or there could have been one customer in the system at time t and the customer did not complete service during Δt . Thus, the following holds:

C. Queueing Theory

$$P_1(t + \Delta t) = P_0(t)P\{1 \text{ arrival during } \Delta t\} + P_1(t)P\{\text{no service completed during } \Delta t\}$$

Because of assumptions (1) and (2), the following probability statements can be used:

$$\begin{aligned} P\{1 \text{ arrival during } \Delta t\} &\cong \lambda \Delta t \\ P\{\text{no arrivals during } \Delta t\} &\cong 1 - \lambda \Delta t \\ P\{\text{service completed during } \Delta t\} &\cong \mu \Delta t \\ P\{\text{no service completed during } \Delta t\} &\cong 1 - \mu \Delta t \end{aligned}$$

This results in the following:

$$\begin{aligned} P_0(t + \Delta t) &= P_0(t)[1 - \lambda \Delta t] + P_1(t)[\mu \Delta t] \\ P_1(t + \Delta t) &= P_0(t)[\lambda \Delta t] + P_1(t)[1 - \mu \Delta t] \end{aligned}$$

Collecting the terms in the equations, rearranging, dividing by Δt , and taking the limit as goes Δt to zero, yields the following set of differential equations:

$$\begin{aligned} \frac{dP_0(t)}{dt} &= \lim_{\Delta t \rightarrow 0} \frac{P_0(t + \Delta t) - P_0(t)}{\Delta t} = -\lambda P_0(t) + \mu P_1(t) \\ \frac{dP_1(t)}{dt} &= \lim_{\Delta t \rightarrow 0} \frac{P_1(t + \Delta t) - P_1(t)}{\Delta t} = \lambda P_0(t) - \mu P_1(t) \end{aligned}$$

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:

$$P_0(t) = \left(\frac{\mu}{\lambda + \mu} \right) + \left(\frac{\lambda}{\lambda + \mu} \right) e^{-(\lambda + \mu)t} \quad (\text{C.1})$$

$$P_1(t) = \left(\frac{\lambda}{\lambda + \mu} \right) - \left(\frac{\lambda}{\lambda + \mu} \right) e^{-(\lambda + \mu)t} \quad (\text{C.2})$$

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:

$$P_0 = \lim_{t \rightarrow \infty} P_0(t) = \frac{\mu}{\lambda + \mu} \quad (\text{C.3})$$

$$P_1 = \lim_{t \rightarrow \infty} P_1(t) = \frac{\lambda}{\lambda + \mu} \quad (\text{C.4})$$

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, both from 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, that the time between arrivals of the customer is exponentially distributed and that 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 C.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 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.

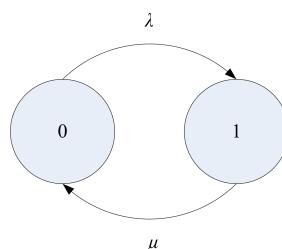


Figure C.4.: Two state rate transition diagram

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, i.e. μP_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, i.e. λP_0 . Using these ideas yields:

State	rate in	=	rate out
0	μP_1	=	λP_0
1	λP_0	=	μP_1

C. Queueing Theory

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 in Equation (C.3) and Equation (C.4). Sets of equations derived in this manner are called *steady state* equations.

Now, more general situations can be examined. Consider a general queueing 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.

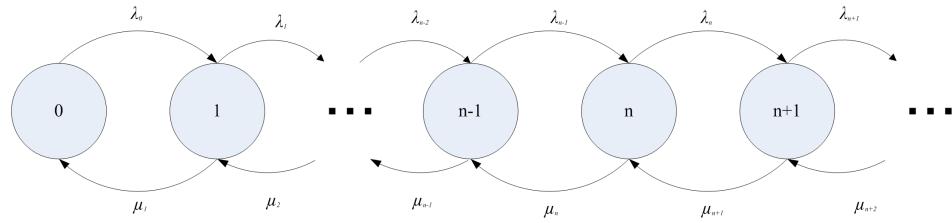


Figure C.5.: General rate transition diagram for any number of states

Figure C.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:

$$P_n = P\{N = n\} = \lim_{t \rightarrow \infty} P\{N(t) = n\}$$

as the steady state probability that there are n customers in the system. Let λ_n be the mean arrival rate of customers entering the system when there are n customers in the system, $\lambda_n \geq 0$. Let μ_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, i.e. $N \in \{0, 1, 2, \dots\}$. The steady state equations for this situation are as follows:

State	rate in	=	rate out
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:

$$P_1 = \frac{\lambda_0}{\mu_1} P_0$$

C.1. Single Line Queueing Stations

$$P_2 = \frac{\lambda_1 \lambda_0}{\mu_2 \mu_1} P_0$$

⋮

$$P_n = \frac{\lambda_{n-1} \lambda_{n-2} \cdots \lambda_0}{\mu_n \mu_{n-1} \cdots \mu_1} P_0 = \prod_{j=1}^{n-1} \left(\frac{\lambda_j}{\mu_{j+1}} \right) P_0$$

for $n = 1, 2, 3, \dots$. Provided that $\sum_{n=0}^{\infty} P_n = 1$, P_0 can be computed as:

$$P_0 = \left[\sum_{n=0}^{\infty} \prod_{j=0}^{n-1} \left(\frac{\lambda_j}{\mu_{j+1}} \right) \right]^{-1}$$

Therefore, for any given set of λ_n and μ_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:

$$L = \sum_{n=0}^{\infty} n P_n$$

$$L_q = \sum_{n=c}^{\infty} (n - c) P_n$$

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, e.g. finite system size, not all customers that arrive will enter the queue. Little's formula is true for the customers that 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:

$$\lambda_e = \sum_{n=0}^{\infty} \lambda_n P_n$$

For finite system size, k , we have:

$$\lambda_e = \sum_{n=0}^{k-1} \lambda_n P_n$$

C. Queueing Theory

since $\lambda_n = 0$ for $n \geq k$. This is because nobody can enter when the system is full. All these relationships yield:

$$\begin{aligned} L &= \lambda_e W \\ L_q &= \lambda_e W_q \\ B &= \frac{\lambda_e}{\mu} \\ \rho &= \frac{\lambda_e}{c\mu} \\ L &= L_q + B \\ W &= W_q + \frac{1}{\mu} \end{aligned}$$

Section C.4 presents the results of applying the general solution for P_n to different queueing system configurations. Table C.3 presents specific results for the M/M/c queuing system for $c = 1, 2, 3$. Using these results and those in Section C.4, the analysis of a variety of different queueing situations is possible.

Table C.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	$\frac{2(1 - \rho)}{2 + 4\rho + 3\rho^2}$	$\frac{9\rho^4}{2 + 2\rho - \rho^2 - 3\rho^3}$

This section has only scratched the surface of queueing theory. A vast amount of literature is available on queueing 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 on-line resources available on the topic. The interested reader should search on “Queueing Theory Books On Line”. The next section presents some simple examples to illustrate the use of the formulas.

C.2. Examples and Applications of Queueing 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. On line resources for software for queueing analysis can be found at:

- List of queueing theory software resources¹
- QTSPlus² Excel based queueing theory software that accompanies Gross et al. (2008)

The most important part of performing a queueing analysis is to identify the most appropriate queueing 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 queueing model and its input parameters
4. Identify the desired performance measures
5. Compute the required performance measures

C.2.1. Infinite Queue Examples

In this section, we will explore two queueing 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 queueing systems.

Example C.1. 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 3 spaces in front of the window, including that for the car being served. Other arriving cars can wait outside these 3 spaces. The pharmacy is interested in answering the following questions:

1. What is the probability that an arriving customer can enter one of the 3 spaces in front of the window?
2. What is the probability that an arriving customer will have to wait outside the 3 spaces?
3. What is the probability that an arriving customer has to wait?
4. How long is an arriving customer expected to wait before starting service?
5. 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?

¹<https://web2.uwindsor.ca/math/hlynka/qsoft.html>

²<http://mason.gmu.edu/~jshortle/fqt5th.html>

Solution to Example C.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 1 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 that arrive when the three spaces are filled, still wait for service outside the 3 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 a M/M/1 queueing situation with an arrival rate $\lambda = 10/\text{hr}$ and a service rate of $\mu = 12/\text{hr}$. Notice the input parameters have been converted to a common unit of measure (customers/hour).

The equations for the M/M/1 can be readily applied. From Section C.4 the following formulas can be applied:

$$\begin{aligned}
 c &= 1 \\
 \lambda_n &= \lambda = 10/\text{hr} \\
 \lambda_e &= \lambda \\
 \mu_n &= \mu = 12/\text{hr} \\
 \rho &= \frac{\lambda}{c\mu} = r = 10/12 = 5/6 \\
 P_0 &= 1 - \frac{\lambda}{\mu} = 1 - r = 1/6 \\
 P_n &= P_0 r^n = \frac{1}{6} \left(\frac{5}{6}\right)^n \\
 L &= \frac{r}{1-r} = \frac{(5/6)}{1-(5/6)} = 5 \\
 L_q &= \frac{r^2}{1-r} = \frac{(5/6)^2}{1-(5/6)} = 4.16 \\
 W_q &= \frac{L_q}{\lambda} = \frac{r}{\mu(1-r)} = 0.416 \text{ hours} = 25.02 \text{ minutes} \\
 W &= \frac{L}{\lambda} = 5/10 = 0.5 \text{ hours} = 30 \text{ minutes}
 \end{aligned}$$

Let's consider each question from the problem in turn:

1. 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 3 spaces in front of the window, you should consider the question: *When can a customer enter one of the 3 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 3 customers in the system, then the 3rd 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 $P[N \leq 2] = 1 - r^3 \cong 0.42$

2. An arriving customer will have to wait outside of the 3 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), $P[N > 2] = 1 - P[N \leq 2] \cong 0.58$
3. An arriving customer has to wait when there are 1 or more customers already at the pharmacy. This is $P[N \geq 1] = 1 - P[N < 1] = 1 - P_0$.

$$P[N \geq 1] = 1 - P[N < 1] = 1 - P_0 = 1 - (1-r) = \rho = \frac{5}{6}$$

4. The waiting time that does not include service is the queueing time. Thus, W_q needs to be computed to answer question (d).

$$W_q = 0.41\bar{6} \text{ hours} = 25.02 \text{ minutes}$$

5. 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 $P\{N \leq m-1\} = 0.4$.

$$\begin{aligned} P[N \leq m-1] &= \gamma \\ 1 - r^m &= \gamma \\ r^m &= 1 - \gamma \\ m = \frac{\ln(1-\gamma)}{\ln r} &= \frac{\ln(1-0.4)}{\ln(\frac{5}{6})} = 2.8 \cong 3 \text{ spaces} \end{aligned}$$

Rounding up guarantees $P[N \leq m-1] \geq 0.4$.

C. Queueing Theory

The following example models self-service copiers.

Example C.2. 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 4 or more people in the copy center to be less than 5%. How many copiers should they install?

Solution to Example C.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 QTSPlus³ software was used.

The software is very self-explanatory. It is important to remember that when applying the queueing formulas make sure that you keep your time units consistent. Setting up the QTSPlus spreadsheet as shown in Figure C.6 with $c = 3$ yields the results shown in Figure C.7. By changing the number of servers, one can find that $c = 3$ meets the probability requirement as shown in Table C.4.

Table C.4.: Results for Example C.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

³<http://mason.gmu.edu/~jshortle/fqt5th.html>

C.2. Examples and Applications of Queueing Analysis

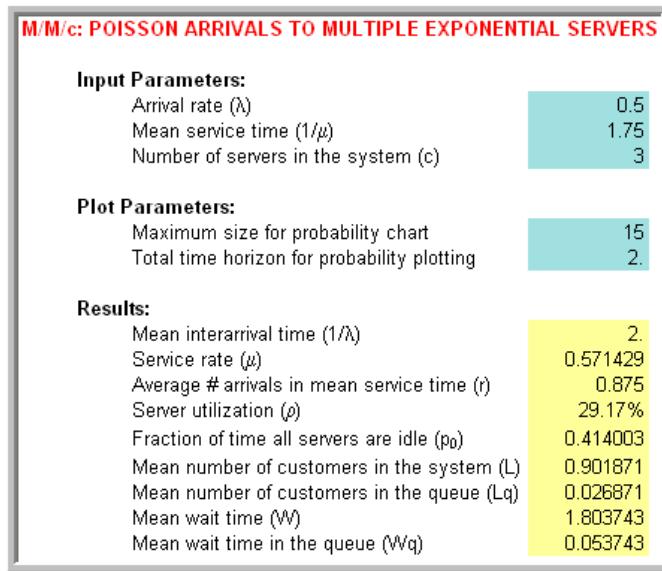


Figure C.6.: QTSPlus M/M/c spreadsheet results

	A	B	C	D
1	Customer Size Distribution			
2	n	prob(n)	CDF(n)	1-CDF(n)
3	0	0.414003	0.414003	0.585997
4	1	0.362253	0.776256	0.223744
5	2	0.158486	0.934741	0.065259
6	3	0.046225	0.980966	0.019034
7	4	0.013482	0.994448	0.005552
8	5	0.003932	0.998381	0.001619
9	6	0.001147	0.999528	0.000472
10	7	0.000335	0.999862	0.000138
11	8	0.000098	0.999960	0.000040
12	9	0.000028	0.999988	0.000012
13	10	0.000008	0.999997	0.000003
14	11	0.000002	0.999999	0.000001
15	12	0.000001	1.000000	0.000000
16	13	0.000000	1.000000	0.000000
17	14	0.000000	1.000000	0.000000
18	15	0.000000	1.000000	0.000000

Figure C.7.: QTSPlus M/M/c state probability results

C. Queueing Theory

C.2.1.1. 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 C.2. Another common design situation is to determine the value c of 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:

$$P_w = P[N \geq c] = \sum_{n=c}^{\infty} P_n = \frac{\frac{r^c}{c!}}{\frac{r^c}{c!} + (1 - \rho) \sum_{j=0}^{c-1} \frac{r^j}{j!}}$$

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.

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 γ_α is the solution to the equation:

$$\frac{\gamma\Phi(\gamma)}{\varphi(\gamma)} = \frac{1 - \alpha}{\alpha}$$

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, α , in the form a probability tolerance, you can find the number of servers that will result in the probability of waiting being less than α . This has very useful application in the area of call service centers and help support lines.

Example C.3. 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.

Solution to Example C.3

For $\lambda = 0.5$ and $\mu = 1/1.75$ per minute, you have that $r = 0.875$. Figure C.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 γ_α .

For this problem, this results in about 3 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.

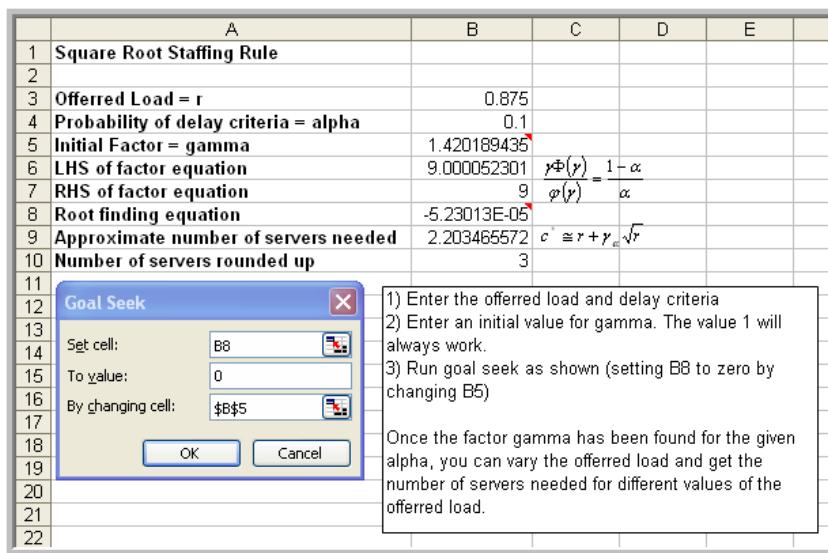


Figure C.8.: Spreadsheet for square root staffing rule

The previous examples have had infinite system capacity. The next section presents examples of finite capacity queueing systems.

C.2.2. Finite Queue Examples

In this section, we will explore three queueing 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 C.4. 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

C. Queueing Theory

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 C.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 service) and $\lambda = 1$ and $\mu = 1.2$ per minute. The desired performance measures are W and L . Figure C.9 presents the results using QTSPlus. Notice that in this case, the effective arrival rate must be computed:

$$\lambda_e = \sum_{n=0}^{k-1} \lambda_n P_n = \sum_{n=0}^{k-1} \lambda P_n = \lambda \sum_{n=0}^{k-1} P_n = \lambda(1 - P_k)$$

Rearranging this formula, yields, $\lambda = \lambda_e + \lambda_f$ where $\lambda_f = \lambda P_k$ equals the mean number of customers that are turned away from the system because it is full. According to Figure C.9, the expected number of turned away because the system is full is about 0.077 per minute (or about 4.62 per hour).

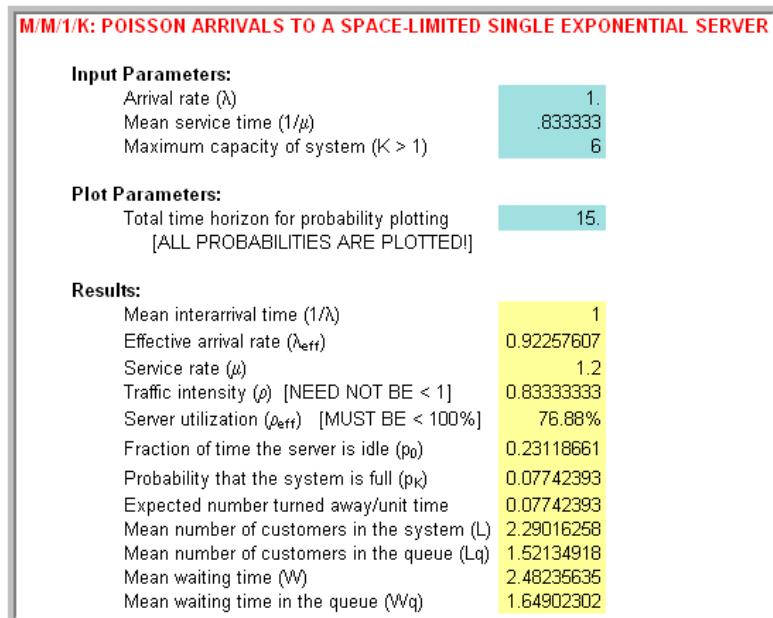


Figure C.9.: Results for finite buffer M/M/1/6 queue

Let's take a look at modeling a situation that we experience everyday, parking lots.

Example C.5. 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, i.e. $\$2$ per hour. How much income does the university potentially lose during peak hours on average because the parking spaces are full?

Solution to Example C.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 a 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:

$$P_c = \frac{\frac{r^c}{c!}}{\sum_{n=0}^c \frac{r^n}{n!}}$$

Customers arrive at the rate $\lambda = 20/hr$ whether the system is full or not. Thus, the expected number of lost customers per hour is λP_c . Since each customer brings $1/\mu$ service time charged at $w = \$2$ per hour, each arriving customer brings w/μ of income on 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 C.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 hours is entered. The number of parking spaces, 10, is the size of the system.

Thus according to Figure C.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 time distribution does not matter. The mean of the service distribution is the critical input parameter to this analysis.

In the next example, a *finite population* of customers that can arrive, depart, and then return is considered. A classic and important example of this type of system is the machine interference

C. Queueing Theory

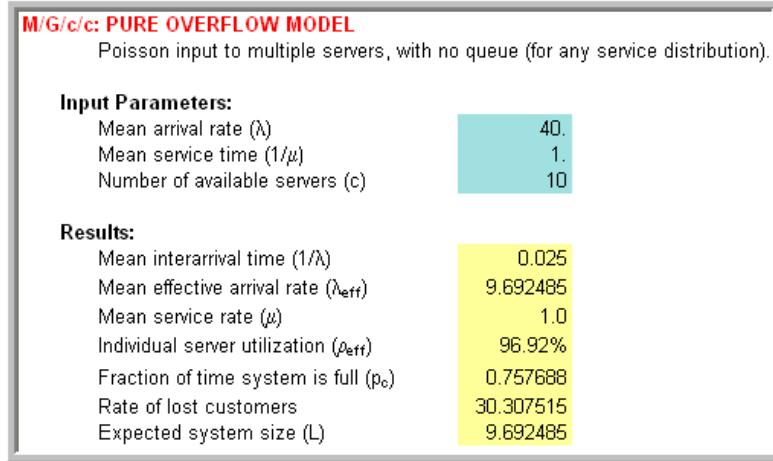


Figure C.10.: Parking lot results for M/M/10/10 system

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 1 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, 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 queueing 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*, λ , is specified. This is not the arrival rate of the population as has been previously utilized. The service rate of μ for each operator is also necessary. The arrival and service rates for this system are:

$$\lambda_n = \begin{cases} (k - n)\lambda & n = 0, 1, 2, \dots k \\ 0 & n \geq k \end{cases}$$

$$\mu_n = \begin{cases} n\mu & n = 1, 2, \dots c \\ c\mu & n \geq c \end{cases}$$

These rates are illustrated in the state diagram of Figure C.11 for the case of 2 operators and 5 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 5λ . This is because there are 5 machines that are not broken down, each with individual rate, λ . Thus, the total rate

of arrivals to the system is 5λ . This rate goes down as more machines breakdown. When all the machines are broken down, the arrival rate to the system will be zero.

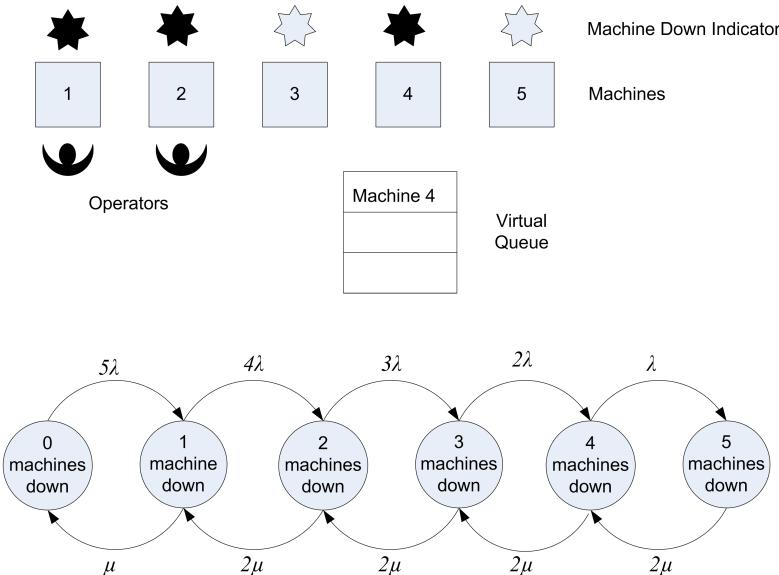


Figure C.11.: System involving machine interference

Notice also that the service rate from state 1 to state 0 is μ . When one machine is in the system, the operator works at rate μ . Note also that the rate from state 2 to state 1 is 2μ . This is because when there are 2 broken down machines the 2 operators are working each at rate μ . Thus, the rate of leaving from state 2 to 1 is 2μ . Because there are only 2 operators in this illustration, the maximum rate is 2μ for state 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's consider an example of this system.

Example C.6. Suppose a manufacturing system contains 5 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 2 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 1 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 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 C.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's 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, e.g. the cost per hour. Clearly, the system costs $15 \times c$ (\$/hour) to employ c operators. The problem also states that it costs \$60/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 queueing system.

In terms of queueing 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:

$$E[TC] = 60 \times L + 15 \times c$$

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 C.12 illustrates the inputs for the case of 1 server with 5 machines. Note that the software requests the time between arrivals, $1/\lambda$ and the mean service time, $1/\mu$, rather than λ and μ .

Table C.5.: Tabulated Results for Example C.6

	0.10	0.10	0.10	0.10	0.10
λ	0.10	0.10	0.10	0.10	0.10
μ	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

C.3. Non-Markovian Queues and Approximations

MARKOV MULTI-SERVER, FINITE-SOURCE QUEUE WITHOUT SPARES
 (Machine repair problem with multiple repairmen)
 After entering input parameters, press "Solve" button.

Input Parameters:	Mean interarrival time for a single customer ($1/\lambda$) Mean time to complete service ($1/\mu$) Number of servers in the system (c) Maximum # of units in the system (M) Specific time for delay distribution calculation (t)	10. (i.e., Mean time between failures) 4. (i.e., Mean time to repair) 1 (i.e., Number of repair persons) 5 (i.e., Number of repairable units) 10.
<input type="button" value="Solve"/>		
Results:	Combined effective overall mean arrival rate (λ_{eff}) 0.2325672 Mean service rate (μ) 0.25 System utilization (ρ_{eff}) 93.03% Fraction of time server is idle (p_0) 0.069731 Expected queue size (L_q) 1.7440589 Expected system size (L) 2.6743278 Expected waiting time in the queue (W_q) 7.4991605 Expected waiting time in the system (W) 11.49916 Probability that an arrival's line delay exceeds t 0.297388	

Figure C.12.: Modeling the machine interference problem in QTSPlus

Table C.5 shows the results of the analysis. The results indicate that as the number of operators increase, the expected cost reaches its minimum at $c = 2$. As the number of 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:

$$\text{Machine Utilization} = \overline{MU} = \frac{k - L}{k} = 1 - \frac{L}{k}$$

C.3. Non-Markovian Queues and Approximations

So far, the queueing 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 case is the M/G/1 queue and the second case is an approximation for the GI/G/c queueing 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 independent and identically distributed random variables with any distribution. Table C.6 presents the basic results for the M/G/1 and M/D/1 queueing systems.

C. Queueing Theory

Table C.6.: Results M/G/1 and M/D/1

Model	Parameters	L_q
M/G/1	$E[ST] = \frac{1}{\mu}; Var[ST] = \sigma^2; r = \lambda/\mu$	$L_q = \frac{\lambda^2 \sigma^2 + r^2}{2(1 - r)}$
M/D/1	$E[ST] = \frac{1}{\mu}; Var[ST] = 0; r = \lambda/\mu$	$L_q = \frac{r^2}{2(1 - r)}$

For the M/G/1 model with a service distribution having a mean $E[ST] = 1/\mu$ and variance σ^2 , the expected number in the system is:

$$L_q = \frac{\lambda^2 \sigma^2 + r^2}{2(1 - r)} + r$$

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:

$$W_q(GI/G/1) \approx \left(\frac{c_a^2 + c_s^2}{2} \right) W_q(M/M/1)$$

In this equation, $W_q(M/M/1)$ denotes the expected waiting time in the queue for the M/M/1 model c_a^2 and c_s^2 and represent the squared coefficient of variation for the inter-arrival time and service time distributions. Recall that for a random variable, X , the squared coefficient of variation is given by $c_X^2 = Var[X]/(E[X])^2$. (Whitt, 1983) used a very similar approximation for the GI/G/c queue to compute the traffic congestion at each node in a queueing network for his Queueing Network Analyzer:

$$W_q(GI/G/c) \approx \left(\frac{c_a^2 + c_s^2}{2} \right) W_q(M/M/c)$$

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 inter-arrival and service time distributions and a way to compute the waiting time in the queue for a M/M/c queueing system. These results are useful when trying to verify and validate a simulation model of a queueing system, especially in the case of a system that consists of

more than one queueing system organized into a network. Before examining that more complicated case, some of the issues related to using to simulate single queue systems should be examined.

The following section summarizes the formulas for the previously mentioned queueing systems. The appendix is finished off with some example exercises that can be used to test your understanding of the application of these formulas.

C.4. Summary of Queueing Formulas

This section provides the formulas for basic single queue stations and is meant simply as a resource where the formulas are readily available for potential application. The following notation is used within this section.

Let N represent the steady state number of customers in the system, where $N \in \{0, 1, 2, \dots, k\}$ where k is the maximum number of customers in the system and may be infinite (∞).

Let λ_n be the arrival rate when there are $N = n$ customers in the system.

Let μ_n be the service rate when there are $N = n$ customers in the system.

Let $P_n = P[N = n]$ be the probability that there are n customers in the system in steady state.

When λ_n is constant for all n , we write $\lambda_n = \lambda$.

When μ_n is constant for all n , we write $\mu_n = \mu$.

Let λ_e be the effective arrival rate for the system, where

$$\lambda_e = \sum_{n=0}^{\infty} \lambda_n P_n$$

Since $\lambda_n = 0$ for $n \geq k$ for a finite system size, k , we have:

$$\lambda_e = \sum_{n=0}^{k-1} \lambda_n P_n$$

Let $\rho = \frac{\lambda}{c\mu}$ be the utilization.

Let $r = \frac{\lambda}{\mu}$ be the offered load.

C. Queueing Theory

C.4.1. M/M/1 Queue

$$\begin{aligned}\lambda_n &= \lambda \\ \mu_n &= \mu \\ r &= \lambda/\mu\end{aligned}$$

$$\begin{aligned}P_0 &= 1 - r \\ P_n &= P_0 r^n \\ L_q &= \frac{r^2}{1 - r}\end{aligned}$$

Table C.7.: Results M/G/1 and M/D/1

Model	Parameters	L_q
M/G/1	$E[ST] = \frac{1}{\mu}; Var[ST] = \sigma^2; r = \lambda/\mu$	$L_q = \frac{\lambda^2 \sigma^2 + r^2}{2(1 - r)}$
M/D/1	$E[ST] = \frac{1}{\mu}; Var[ST] = 0; r = \lambda/\mu$	$L_q = \frac{r^2}{2(1 - r)}$

C.4.2. M/M/c Queue

$$\begin{aligned}\lambda_n &= \lambda \\ \mu_n &= \begin{cases} n\mu & 0 \leq n < c \\ c\mu & n \geq c \end{cases} \\ \rho &= \lambda/c\mu \quad r = \lambda/\mu\end{aligned}$$

$$\begin{aligned}P_0 &= \left[\sum_{n=0}^{c-1} \frac{r^n}{n!} + \frac{r^c}{c!(1-\rho)} \right]^{-1} \\ L_q &= \left(\frac{r^c \rho}{c!(1-\rho)^2} \right) P_0\end{aligned}$$

$$P_n = \begin{cases} \frac{(r^n)^2}{n!} P_0 & 1 \leq n < c \\ \frac{r^n}{c! c^{n-c}} P_0 & n \geq c \end{cases}$$

Table C.8.: 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	$\frac{2(1 - \rho)}{2 + 4\rho + 3\rho^2}$	$\frac{9\rho^4}{2 + 2\rho - \rho^2 - 3\rho^3}$

C.4.3. M/M/c/k Queue

$$\begin{aligned}\lambda_n &= \begin{cases} \lambda & n < k \\ 0 & n \geq k \end{cases} \\ \mu_n &= \begin{cases} n\mu & 0 \leq n < c \\ c\mu & c \leq n \leq k \end{cases} \\ \rho &= \lambda/c\mu \quad r = \lambda/\mu \\ \lambda_e &= \lambda(1 - P_k) \\ P_0 &= \begin{cases} \left[\sum_{n=0}^{c-1} \frac{r^n}{n!} + \frac{r^c}{c!} \frac{1 - \rho^{k-c+1}}{1 - \rho} \right]^{-1} & \rho \neq 1 \\ \left[\sum_{n=0}^{c-1} \frac{r^n}{n!} + \frac{r^c}{c!} (k - c + 1) \right]^{-1} & \rho = 1 \end{cases} \\ P_n &= \begin{cases} \frac{r^n}{n!} P_0 & 1 \leq n < c \\ \frac{r^n}{c! c^{n-c}} P_0 & c \leq n \leq k \end{cases}\end{aligned}$$

C.4.4. M/G/c/c Queue

$$\begin{aligned}\lambda_n &= \begin{cases} \lambda & n < c \\ 0 & n \geq c \end{cases} \\ \mu_n &= \begin{cases} n\mu & 0 \leq n \leq c \\ 0 & n > c \end{cases} \\ \rho &= \lambda/c\mu \quad r = \lambda/\mu \\ \lambda_e &= \lambda(1 - P_k)\end{aligned}$$

C. Queueing Theory

$$P_0 = \left[\sum_{n=0}^c \frac{r^n}{n!} \right]^{-1}$$

$$P_n = \frac{r^n}{n!} P_0$$

$$0 \leq n \leq c$$

$$L_q = 0$$

C.4.5. M/M/1/k Queue

$$\lambda_n = \begin{cases} (k-n)\lambda & 0 \leq n < k \\ 0 & n \geq k \end{cases}$$

$$\mu_n = \begin{cases} (k-n)\lambda & 0 \leq n \leq k \\ 0 & n > k \end{cases}$$

$$r = \lambda/\mu \quad \lambda_e = \lambda(k-L)$$

$$P_0 = \left[\sum_{n=0}^k \prod_{j=0}^{n-1} \left(\frac{\lambda_j}{\mu_{j+1}} \right) \right]^{-1}$$

$$P_n = \binom{k}{n} n! r^n P_0$$

$$L_q = \begin{cases} \frac{\rho}{1-\rho} - \frac{\rho(k\rho^k + 1)}{1-\rho^{k+1}} & \rho \neq 1 \\ \frac{k(k-1)}{2(k+1)} & \rho = 1 \end{cases}$$

C.4.6. M/M/c/k Queue

$$\lambda_n = \begin{cases} (k-n)\lambda & 0 \leq n < k \\ 0 & n \geq k \end{cases}$$

$$\mu_n = \begin{cases} n\mu & 0 \leq n < c \\ c\mu & n \geq c \end{cases}$$

$$r = \lambda/\mu \quad \lambda_e = \lambda(k-L)$$

$$P_0 = \left[\sum_{n=0}^k \prod_{j=0}^{n-1} \left(\frac{\lambda_j}{\mu_{j+1}} \right) \right]^{-1}$$

$$P_n = \begin{cases} \binom{k}{n} r^n P_0 & 1 \leq n < c \\ \binom{k}{n} \frac{n!}{c^{n-c} c!} r^n P_0 & c \leq n \leq k \end{cases}$$

$$L_q = \begin{cases} \frac{P_0 r^c \rho}{c!(1-\rho)^2} [1 - \rho^{k-c} - (k-c)\rho^{k-c}(1-\rho)] & \rho < 1 \\ \frac{r^c (k-c)(k-c+1)}{2c!} P_0 & \rho = 1 \end{cases}$$

C.4.7. M/M/1/k/k Queue

$$\lambda_n = \begin{cases} (k-n)\lambda & 0 \leq n < k \\ 0 & n \geq k \end{cases}$$

$$\mu_n = \begin{cases} (k-n)\lambda & 0 \leq n \leq k \\ 0 & n > k \end{cases}$$

$$r = \lambda/\mu \quad \lambda_e = \lambda(k-L)$$

$$P_0 = \left[\sum_{n=0}^k \prod_{j=0}^{n-1} \left(\frac{\lambda_j}{\mu_{j+1}} \right) \right]^{-1}$$

$$P_n = \binom{k}{n} n! r^n P_0$$

$$0 \leq n \leq k$$

$$L_q = k - \left(\frac{\lambda + \mu}{\lambda} \right) (1 - P_0)$$

C.4.8. M/M/c/k/k Queue

$$\lambda_n = \begin{cases} (k-n)\lambda & 0 \leq n < k \\ 0 & n \geq k \end{cases}$$

$$\mu_n = \begin{cases} n\mu & 0 \leq n < c \\ c\mu & n \geq c \end{cases}$$

$$r = \lambda/\mu \quad \lambda_e = \lambda(k-L)$$

$$P_0 = \left[\sum_{n=0}^k \prod_{j=0}^{n-1} \left(\frac{\lambda_j}{\mu_{j+1}} \right) \right]^{-1}$$

C. Queueing Theory

$$P_n = \begin{cases} \binom{k}{n} r^n P_0 & 1 \leq n < c \\ \binom{k}{n} \frac{n!}{c^{n-c} c!} r^n P_0 & c \leq n \leq k \end{cases}$$

$$L_q = \sum_{n=c}^k (n - c) P_n$$

C.5. Exercises

For the exercises in this section, first start with specifying the appropriate queueing models needed to solve the exercise using Kendall's notation. Then, specify the parameters of the model, e.g. λ_e , μ , c , size of the population, size of the system, etc. Specify how and what you would compute to solve the problem. Be as specific as possible by specifying the equations needed. Then, compute the quantities if requested. You might also try to use to solve the problems via simulation.

Exercise C.1. *True or False:* In a queueing 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 queueing.

Exercise C.2. The Burger Joint 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 5,000. On average, how long do potatoes stay in the restaurant before being used? What queuing concept is use to solve this problem?

Exercise C.3. 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 queueing 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?
-

C. Queueing Theory

Exercise C.4. suppose an operator has been assigned to the responsibility of maintaining 3 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 queueing 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?
-

Exercise C.5. 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 queueing theory to SuperFastCopy.

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 inter-arrival 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, 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.

What is the probability that a passenger will have to wait before being screened? On average, how many passengers are waiting in line to enter the checkpoint? On average, how long will a passenger spend at the checkpoint?

Exercise C.6. 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 queueing theory, which machine is better in terms of the average waiting time of the jobs?

Exercise C.7. 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 3 spaces in front of the window including that for the car being served. Other arriving cars can wait outside these 3 spaces. Specify the appropriate queueing 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 3 spaces in front of the window?
 - b. What is the probability that an arriving customer will have to wait outside the 3 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%.
-

Exercise C.8. Joe Rose is a student at Big State U. He does odd jobs to supplement his income. Job requests come every 5 days on the 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?
-

Exercise C.9. 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 the 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. Inter-arrival and service times can be assumed to be exponentially distributed.

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)? If the system has 3 tellers, what is the probability that there will be no one in the bank? What is the expected total cost of the system per hour, when there are 2 tellers?

C. Queueing Theory

Exercise C.10. You have been hired to analyze the needs for loading dock facilities at a trucking terminal. The present terminal has 4 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 5 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 two 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 8 trucks. Then, only when the holding area is full will the trucks be diverted to secondary terminal.

What queuing model should you use to analyze the first proposal? State the model and its parameters. 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. What model should you use to analyze the 2nd proposal? State the model and its parameters.

Exercise C.11. 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 queueing notation, exactly what you would compute to find the percentage of cars that will seek business elsewhere?
 - b. Specify using queueing notation, exactly what you would compute to find the utilization of the pumps?
-

Exercise C.12. 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.

- a. Specify using queueing notation, exactly what you would compute to find the probability of losing a potential customer?

- b. 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?
-

Exercise C.13. 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.

- a. Let the state of the system be the number of machines not working. Construct a state transition diagram for this queueing system.
- b. Write an expression using queueing performance measures to compute the expected downtime cost per week.
-

Exercise C.14. NWH Cardiac Care Unit (CCU) has 5 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 two 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.

- a. What would you compute to find the average number of patients being attended by the nurses?
- b. What would you compute to find the average time that a patient spends waiting for one of the nurses to arrive?
-

Exercise C.15. 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 one mechanic in the company shop.

C. Queueing Theory

Specify using Kendall's notation the correct queueing model for this situation including the appropriate parameters. What would you compute to determine the expected number of repairs that are outsourced per week?

Exercise C.16. 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 the 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 2 chairs available with no waiting room in the shop.

- a. Let the state of the system be the number of customers in the shop, Construct a state transition diagram for this queueing system.
 - b. What is the probability that a customer is turned away?
 - c. What is the probability that the barber shop is idle?
 - d. What is the steady-state mean number of customers in the shop?
-

Exercise C.17. 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, $\bar{N}(t)$, 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

- a. Give a formula for estimating the time average number in the system, $\bar{N}(t)$, and then use the data to compute the time average number 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.
 - d. What queueing formula relationship is used in this problem?
-

D. Miscellaneous Topics in Arena

This appendix covers a number of miscellaneous topics that may be useful when using Arena as your primary simulation modeling environment. The appendix starts with presenting an overview of how to get help in Arena and on utilizing the Arena Run Controller. In addition, the appendix overviews some of the programming aspects of using the Arena modeling environment. The appendix also covers the topic of resource and entity costing within the context of Arena's activity based costing constructs. I have attempted to order the topics in this appendix based on their usefulness within my use of Arena. The next section presents probably the most useful topic: how to get help.

D.1. 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 hyper-links to important information. In fact, as can be seen in Figure D.1, Arena 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 . This text uses a number of Arena's example models (as per Figure D.2) 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 Help system. In addition to the on-line help system, the 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.

The following section provides an introduction to SIMAN for the primary purpose of helping you to understand potential error messages that you might get when developing and debugging an Arena model.

D.2. 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 task. Programming in Arena is quite different than programming in a base language like C. An Arena simulation

D. Miscellaneous Topics in Arena

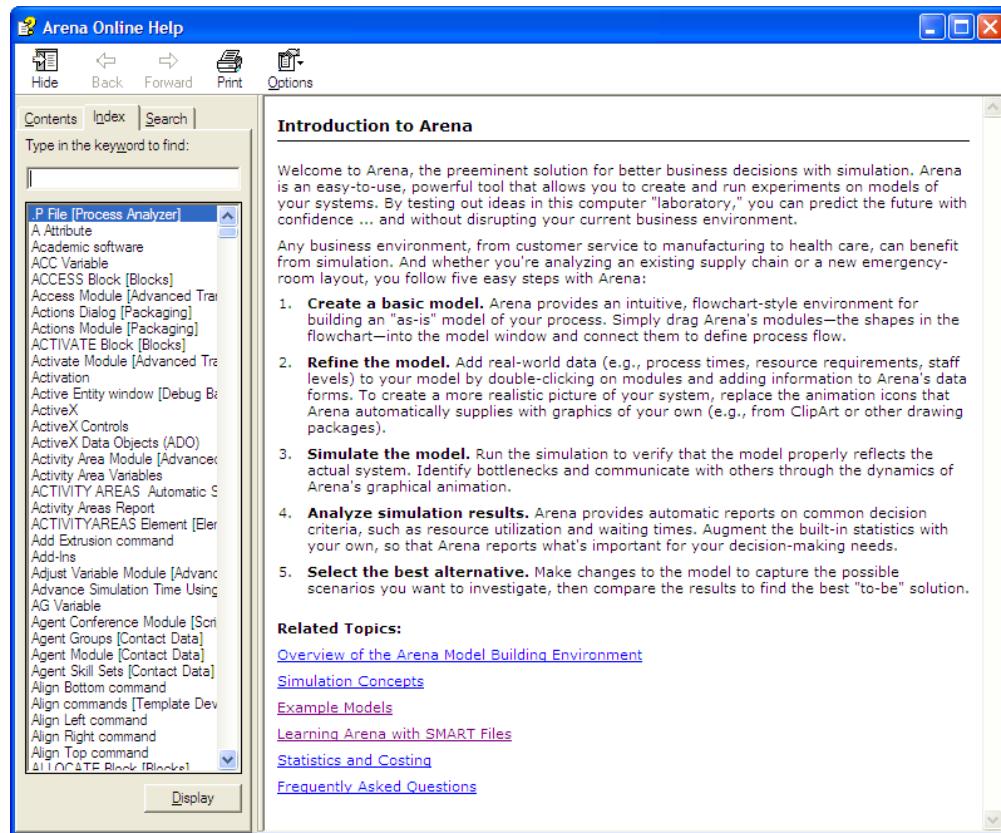


Figure D.1.: Arena help system

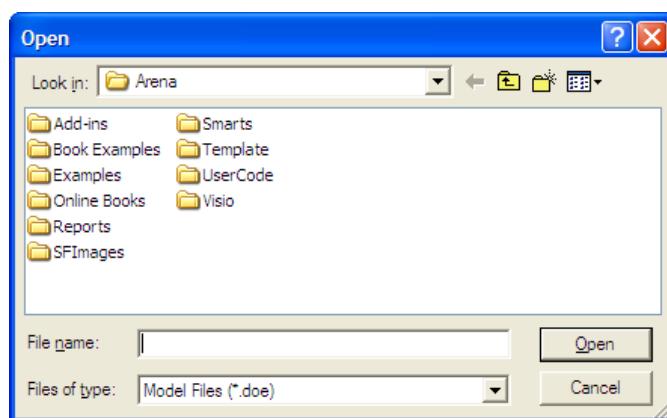


Figure D.2.: Arena folder structure

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 programs, you must have, at the very least, a rudimentary understanding of Arena's underlying language called SIMAN. SIMAN code is produced by the environment, compiled, and then executed. This section provides an overview of SIMAN and Arena's debugger called the Run Controller.

D.2.1. SIMAN MOD and EXP Files

To better understand some of the underlying programming concepts within Arena, it is useful to know that Arena is built on top of the SIMAN simulation programming system. To learn more about SIMAN, I suggest reviewing the following two textbooks 1 and Banks et al. (1995).

The section will review the pharmacy model of Chapter 2 in order to provide an overview of SIMAN. You should open up the pharmacy model from Chapter 2 in order to follow along. Using the Run/SIMAN/View menu will generate two files associated with the SIMAN program, the *mod* (model) file and the *exp* (experiment) file. The files can be viewed using the Window menu (Figure D.3, within the Arena Environment.

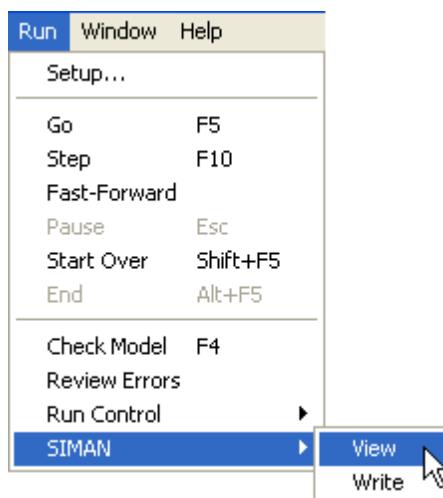


Figure D.3.: Run SIMAN view menu

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.

The following code listing presents the experiment (*exp*) file for the pharmacy model. For readers that 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

D. Miscellaneous Topics in Arena

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 TALIES and OUTPUTS. Each of these constructs will be discussed as you learn more about the modules within the Environment. The experiment module declares and defines that certain elements will be used in the model. For example, the RESOURCES element indicates that there is a resource called Pharmacist that can be 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 Environment.

```
PROJECT,      "Unnamed Project", " ",,,No,Yes,Yes,Yes,No,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,AUTOSTATS(Yes,,);

```

The following listing 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, i.e. *variable*=*expression*.

```

;
;      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;

```

There are two keywords that you should immediately recognize from the model window, CREATE and DISPOSE. Each non-commented line has a line number that identifies the SIMAN

D. Miscellaneous Topics in Arena

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, e.g. 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 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. 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 , the first event is determined by the CREATE module with the smallest “First Creation” time. Figure D.4 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 six minutes to be used as the time of the event associated with the creation of an entity from this CREATE module.

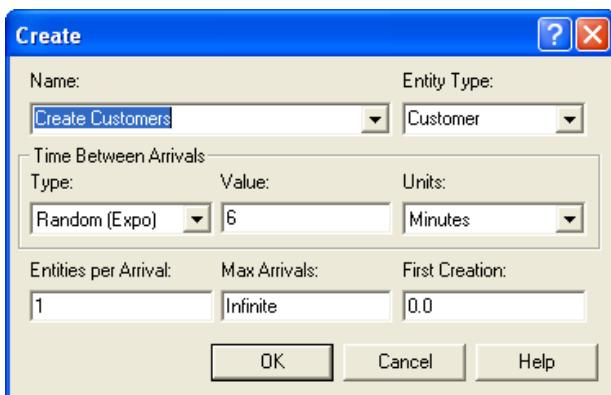


Figure D.4.: CREATE module dialog

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 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. Well, this is a simplified discussion. In general, there are other ways to get an initial event scheduled.

Now that we know that there is a language underneath Arena, we can better use Arena's debugging tool: the Run Controller. The next section provides an overview of using the run controller.

D.2.2. Using the Run Controller

An integral part of any programming effort is debugging the program. Simulation program development 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 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, in this section concentrates on using Arena's Run Controller. Arena's Run Controller is found on the Run menu as shown in Figure D.5.

Figure D.6 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's first learn about the various commands and capabilities of the run controller. Go to 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 D.7.

The Run Controller allows the 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 will see a list of commands for the run controller as shown in Figure D.8. 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 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.

D. Miscellaneous Topics in Arena

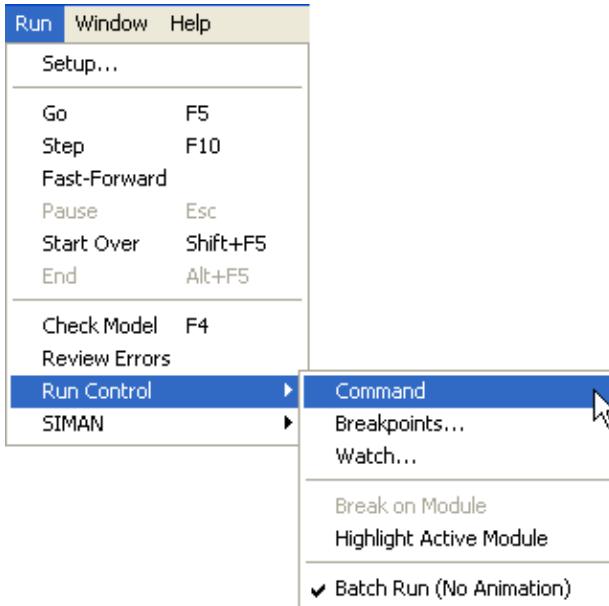


Figure D.5.: Invoking the Run Controller

Now, you should invoke the run controller as indicated in Figure D.5. This will start the run of the model. It will immediately pause the execution of the model at the first execution step. You should see a window as illustrated in Figure D.6. Let's first turn the tracing on for the model. By default, embeds tracing output into the SIMAN code that is generated. Information about the current module and the active entity will be displayed. To turn on the tracing, enter "set trace" at the prompt (in Figure D.9) 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 D.10. You should single step again into the model. Your command window should look like Figure D.10. Remember that 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 D.11.

From this trace output, it is apparent that the entity was sent directly to the SEIZE block from the QUEUE block where it seized 1 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 zero, arrived to the system, and attempted to get a unit of the phar-

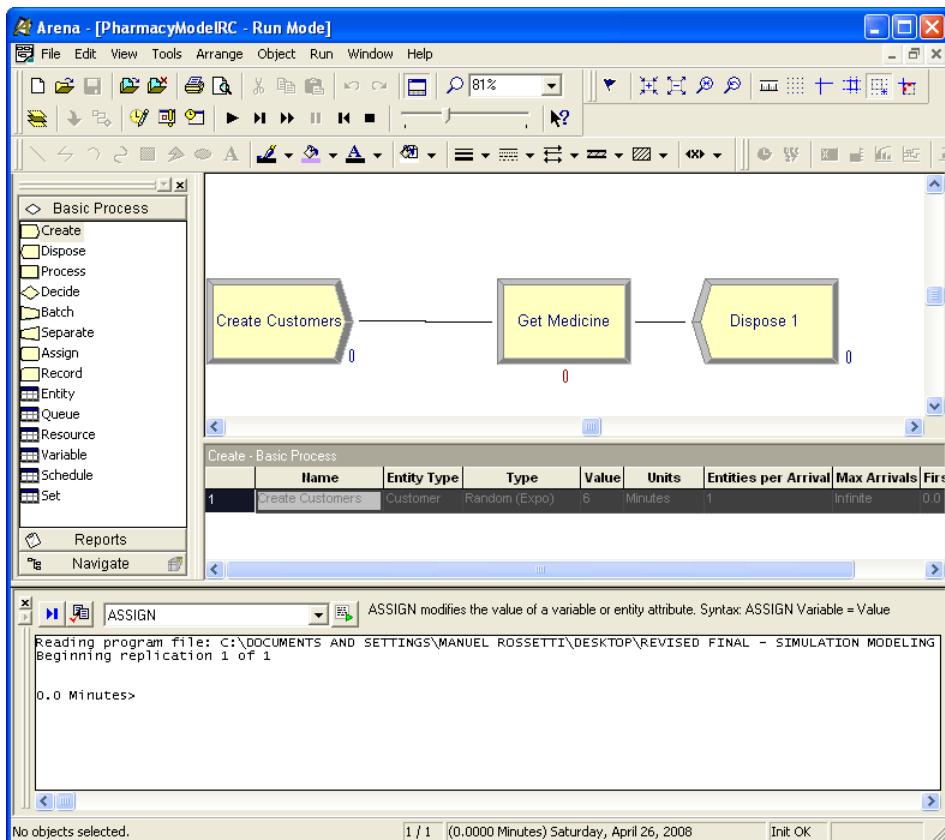


Figure D.6.: Arena Environment with Run Controller invoked

macist. 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 by the DELAY module and is scheduled to "wake up" at time 2.2261054. By using the VIEW CALENDAR command, you can see all the events that are scheduled on the event calendar. Figure D.12 illustrates the output of the VIEW CALENDAR command. There are two data structures within 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 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. uses an internal entity to schedule the event that represents the end of a replication. If there are a large

D. Miscellaneous Topics in Arena

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 D.7.: Getting help for the Run Controller

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 2 are waiting. You can set a watch on the number of customers in the queue and this will cause 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 D.13.

If you type `GO` at the run controller prompt, will run until the watch condition is met, see Figure

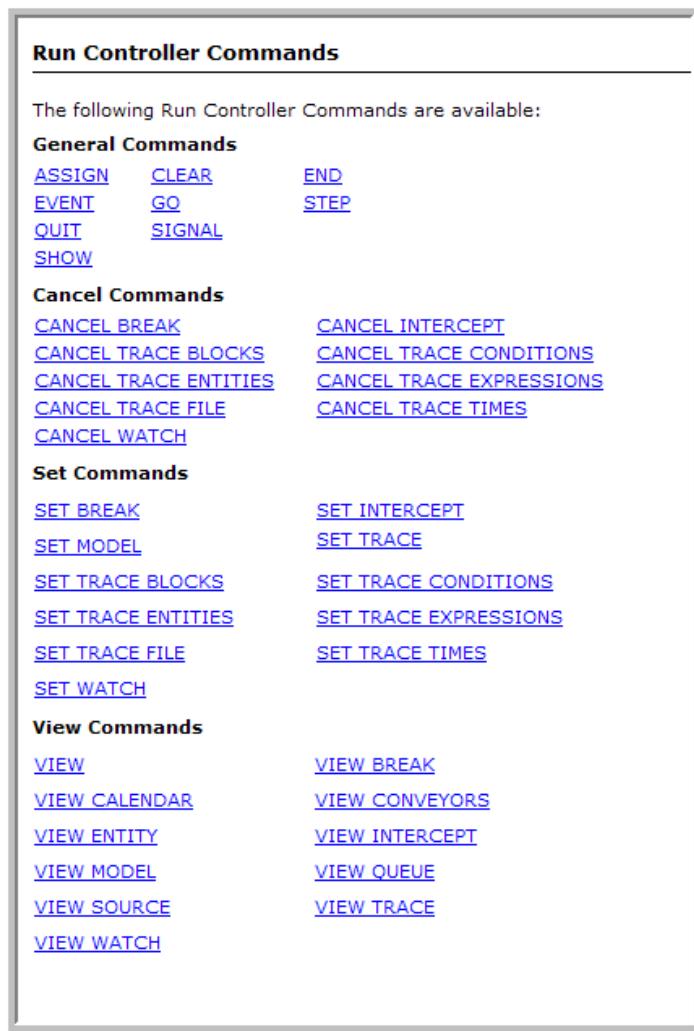


Figure D.8.: The Run Controller commands

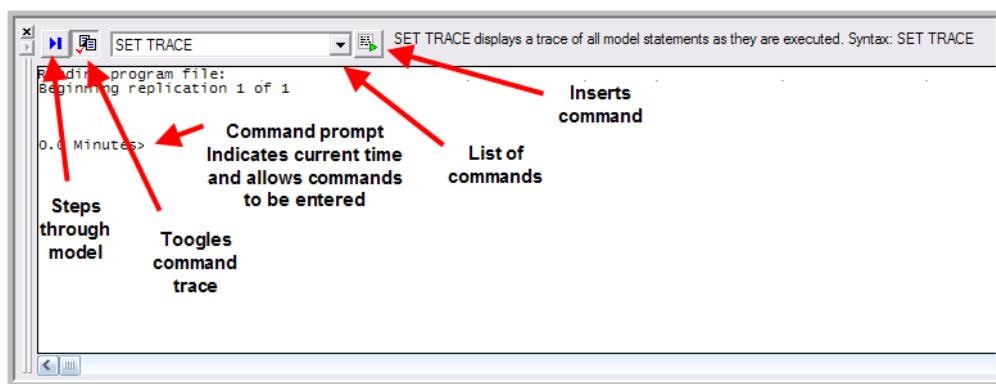


Figure D.9.: Using commands in the Run Controller

D. Miscellaneous Topics in Arena

```

SET TRACE displays a trace of all model statements as they are executed. Syntax: 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.
* 1 2$ CREATE,1,MinutesToBaseTime(0.0),Customer:
MinutesToBaseTime(EXPO(6));
NEXT(3$);

0.0 Minutes>STEP
1 2$ CREATE Entity Type set to Customer
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 D.10.: Run Controller output window after 2 steps

```

SET TRACE displays a trace of all model statements as they are executed. Syntax: SET TRACE
4 9$ 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 D.11.: Run Controller output window after 7 steps

```

*** 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.VATime = 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
2.0769136 Minutes>

```

Figure D.12.: VIEW CALENDAR output

```

*** 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.VATime = 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
2.0769136 Minutes>SET WATCH NQ(Get Medicine.Queue) == 2
Set watch Expression:
Expr# Stop Entity Value Expression
  1   Y      FALSE   NQ(Get Medicine.Queue) == 2
2.0769136 Minutes>

```

Figure D.13.: Setting a WATCH on a variable

D.14. The VIEW QUEUE command will show all the entities in all the queues, see Figure D.15. 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 D.6. 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 D.16. In the figure, the active entity tab is open to allow the attributes of the entity to be seen. The functionality of the debug bar is similar to many debuggers. The 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 com-

D. Miscellaneous Topics in Arena

```

SET WATCH causes the execution of the simulation to be suspended when the value of an expression involving system variables or attributes changes.....
3 0$      ASSIGN      Get Medicine.NumberIn set to 11.0
4 9$      QUEUE       Entity 12 sent to next block
5 8$      SEIZE       Could not seize resource Pharmacist
Entity 12 added to queue Get Medicine.Queue at rank 1
Time: 72.107107 Entity: 13
1 2$      CREATE      Entity Type set to Customer
Next creation scheduled at time 85.253034
Batch of 1 Customer entities created
2 3$      ASSIGN      Create Customers.NumberOut set to 12.0
3 0$      ASSIGN      Get Medicine.NumberIn set to 12.0
Get Medicine.WIP set to 3.0
4 9$      QUEUE       Entity 13 sent to next block
5 8$      SEIZE       Entity 13 added to queue Get Medicine.Queue at rank 2
*** NQ(Get Medicine.Queue) == 2
Changed value at time 72.107107
Old value = FALSE          New Value = TRUE
72.107107 Minutes>
    
```

Figure D.14.: Output after WATCH condition break

```

SET WATCH causes the execution of the simulation to be suspended when the value of an expression involving system variables or attributes changes.....
Changed value at time 72.107107
Old Value = FALSE          New Value = TRUE
72.107107 Minutes>VIEW QUEUE
*** Queue 1 contains 2 Entities ***
*** Rank 1: Entity number 12
Entity.SerialNumber = 12
Entity.Type = 1
Entity.Picture = 11
Entity.Station (M) = 0
Entity.Sequence (NS) = 0
Entity.JobStep (JS) = 0
Entity.CurrentStation = 0
Entity.PlannedStation = 0
Entity.CreateTime = 58.751198
Entity.VATime = 0.0
Entity.NVATime = 0.0
Entity.WaitTime = 0.0
Entity.TranTime = 0.0
Entity.OtherTime = 0.0
*** Rank 2: Entity number 13
Entity.SerialNumber = 13
Entity.Type = 1
Entity.Picture = 11
Entity.Station (M) = 0
Entity.Sequence (NS) = 0
Entity.JobStep (JS) = 0
Entity.CurrentStation = 0
Entity.PlannedStation = 0
Entity.CreateTime = 72.107107
Entity.VATime = 0.0
Entity.NVATime = 0.0
Entity.WaitTime = 0.0
Entity.TranTime = 0.0
Entity.OtherTime = 0.0
72.107107 Minutes>
    
```

Figure D.15.: Output after VIEW QUEUE

Attribute Name	Value
Entity 11	Serial Number: 10
Attributes	
Serial Number	10
Type	1
Animation Picture	11
Station Location	0
Sequence	0
JobStep In Sequence	0
Current Station Location	<None>
Next Planned Station	<None>
Times	
Creation Time	58.553417
Start Time	0.000000
Total VA Time	18.264469
Total NVA Time	0.000000
Total Wait Time	0.000000
Total Transfer Time	0.000000
Total Other Time	0.000000
Group Members	0

View Breakpoints tab View Calendar tab Set variable watch tab

Set breakpoints tab Active Entity tab

72.107107 Minutes>STEP
Time: 76.817885 Entity: 11
SIMAN Run Controller.
* 7 6\$ RELEASE:Pharmacist,1;
76.817885 Minutes>

Figure D.16.: Debug window within the Run Controller

mands 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 using 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. Don't forget to turn the tracing off and to re-save!

The following section introduces some additional programming concepts that may prove useful when building Arena models.

D.3. Programming Concepts within Arena

Within Arena , programming support comes in two forms: laying down flow chart modules and computer language integration (e.g. VBA, C, etc.). 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 4 will be continued. Finally, the use of Visual Basic for Applications (VBA) within the environment will be introduced.

D.3.1. Using the Generated Access File

Each time the simulation experiment is executed 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

D. Miscellaneous Topics in Arena

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 4 within this section to illustrate these concepts.

When you create and run a model, e.g. *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 re-run 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.

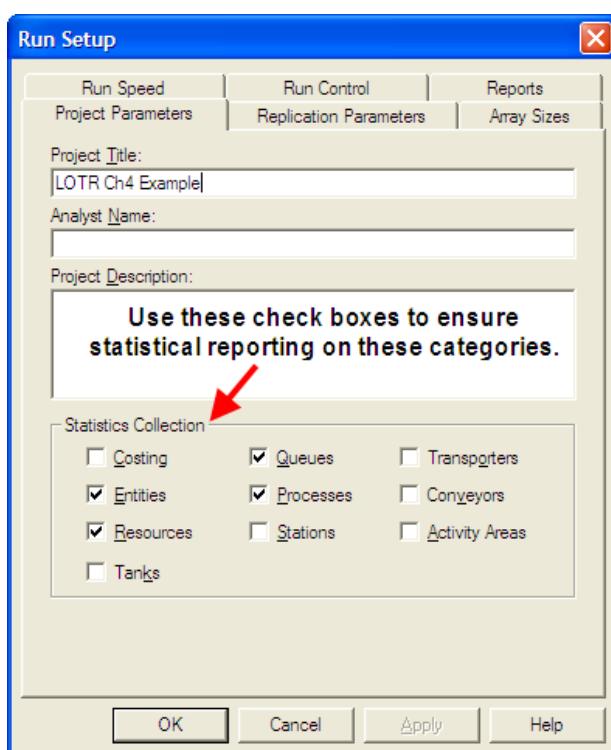


Figure D.17.: Run Setup statistical collection options

The second approach is to use the Projects Parameters panel within the Run > Setup menu dialog as illustrated in Figure D.17. 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's take a look at the type of information that is

stored within the database.

The Arena 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:

Definition Holds the name of the statistical item from within the model, its type (e.g. DSTAT, TALLY, FREQ, etc.) 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 D.18 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, `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).

Definition : Table		
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 ie:DSTAT,CSTAT, TALLY foreign key to Definitiontypes
Limit	Number	

Figure D.18.: Definition table field design

The Statistic table holds the statistical values for the within replication statistics as shown in Figure D.19. 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 *DB-Stat-Example.mdb*. This file was produced by running the *LOTRExample.doe* with 56 replications and renaming the created

D. Miscellaneous Topics in Arena

The screenshot shows a Microsoft Excel-like dialog box titled "Statistic : Table". It contains a table with the following data:

	Field Name	Data Type	Description
1	ID	AutoNumber	
2	ReplicationID	Number	
3	DefinitionID	Number	
4	MinObs	Number	The minimum observation value within this scenario/replicaiton.
5	MaxObs	Number	The maximum observation value within this scenario/replicaiton.
6	AvgObs	Number	The average observation value for this scenario/replicaiton.
7	HalfWidth	Number	The .95 half width for this scenario/ replication.
8	LastValue	Number	The final variable value for this scenario/replicaiton.
9	NumObs	Number	The number of observations in this stat
10	StdDev	Number	The standard deviation of observations

Figure D.19.: Statistic table field

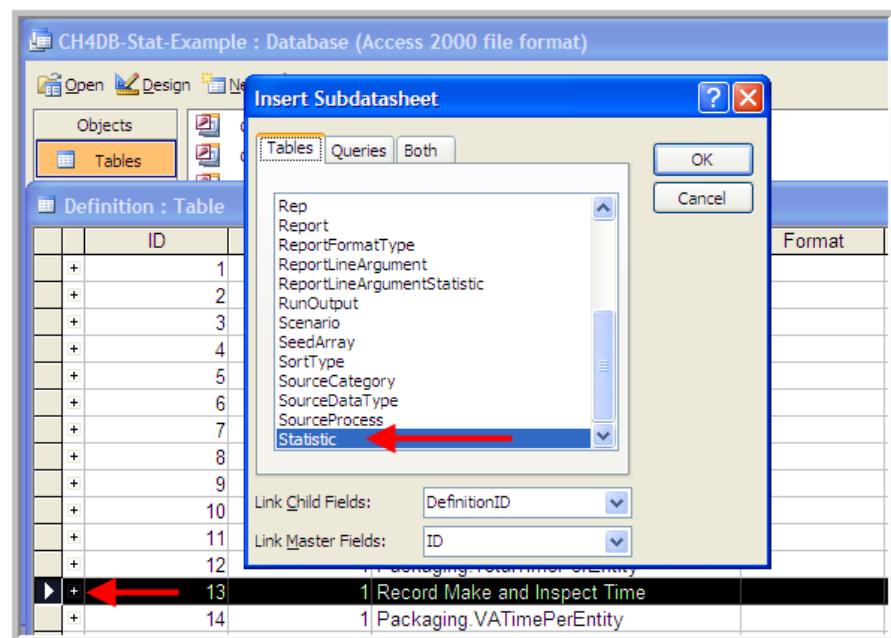


Figure D.20.: Accessing statistics information

database file. If you have Microsoft Access then open the database and then open the table called Definition. Select the desired row, e.g. row 8, and use Insert > Subdatasheet to get the Insert Subdatasheet dialog as shown in Figure D.20. Then, select the Statistic table and press the OK button as shown in the figure.

Figure D.21 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

D.3. Programming Concepts within Arena

The screenshot shows a Microsoft Access-style table named 'Definition : Table'. The table has columns: ID, ReportID, Name, Format, ReportLineDefin, ArgumentIndex, ID, ReplicationID, MinObs, MaxObs, AvgObs, HalfWidth, LastValue. There are two main rows at the top: one for 'Packaging.TotalTimePerEntity' and another for 'Record Make and Inspect Time'. Below these are 56 data rows. Row 13 contains values: ID=13, ReplicationID=1, MinObs=11.4505300186, MaxObs=705.241399798, AvgObs=363.804363517, HalfWidth=2E+20, LastValue=705.241399798. Row 400 contains values: ID=400, ReplicationID=10, MinObs=15.9273709035, MaxObs=806.236078717, AvgObs=402.755368380, HalfWidth=2E+20, LastValue=806.236078717.

Figure D.21.: Replication statistics for make and inspect time

familiar with writing queries, this same information can be extracted by writing the query as shown in Figure D.22.

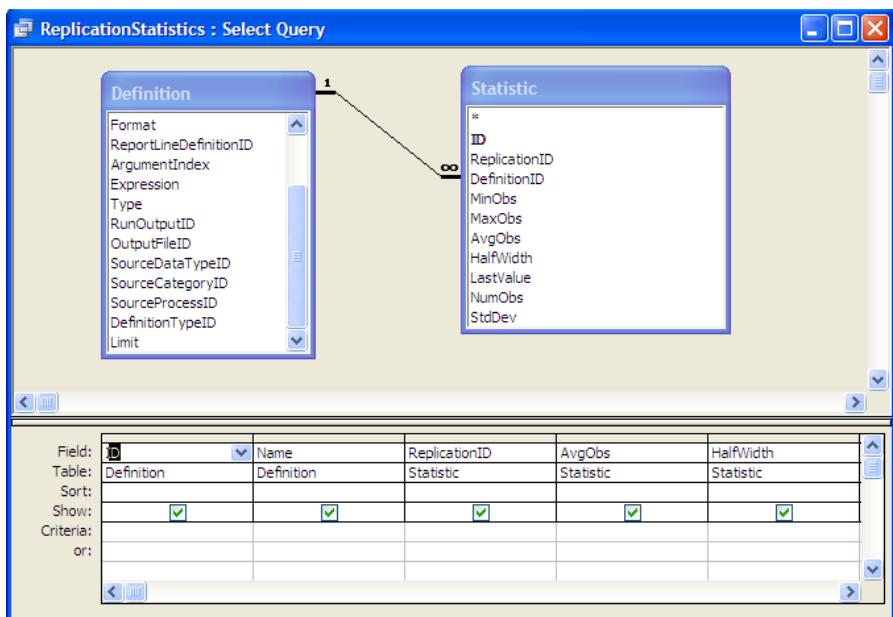


Figure D.22.: Query to access replication statistics

To summarize the statistics across the replications, you can write a group by query as shown in Figure D.23.

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 D.24.

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"

D. Miscellaneous Topics in Arena

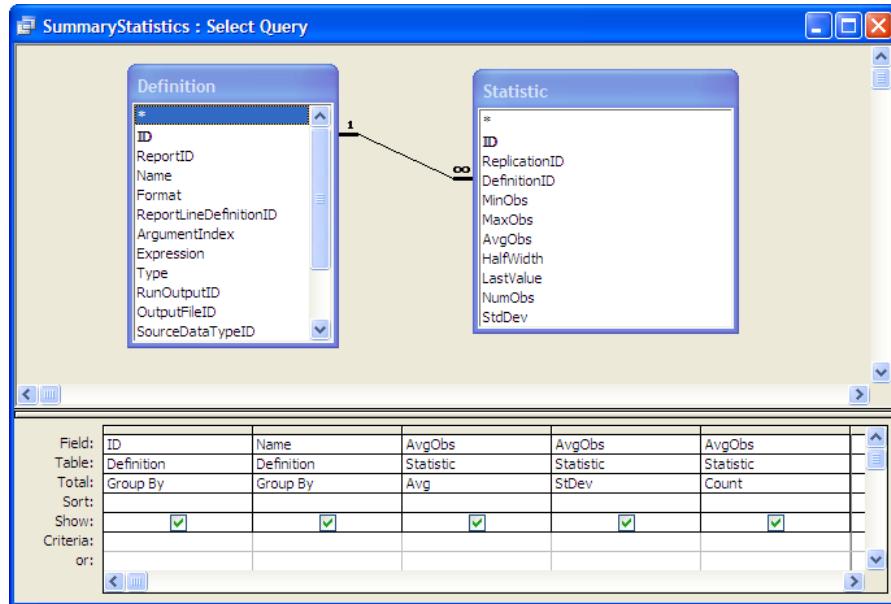


Figure D.23.: Query to summarize replication statistics

	Field Name	Data Type	Description
1	ID	AutoNumber	
2	DefinitionID	Number	Base definition for this counter
3	ReplicationID	Number	The replication number for this count data.
4	Value	Number	The current output value for this scenario/replication.

Figure D.24.: Output table field design

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 D.25. 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 an experienced Microsoft Access user you can then form custom queries, reports, charts, etc. for your simulation results.

47	1	Make Ring Process Accun
48	1	TimeToMakeOrderStat
	ID	ReplicationID
	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 D.25.: Expanded data sheet view for OUTPUT statistics

D.3.2. Working with Files, Excel, and Access

As discussed in Chapter 2, 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 File Name and an optionally a file format. The 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 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 2, 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.

D.3.2.1. Reading from a Text File

In this example, the SMART file, *Smarts162.doe*, is used to show how to read from a text file. Open up the file named *Smarts162Revised.doe*. Figure D.26 provides an overview of the model.

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. The first number in the file is the type (1 or 2). Then, the following two numbers are the station 1 and

D. Miscellaneous Topics in Arena

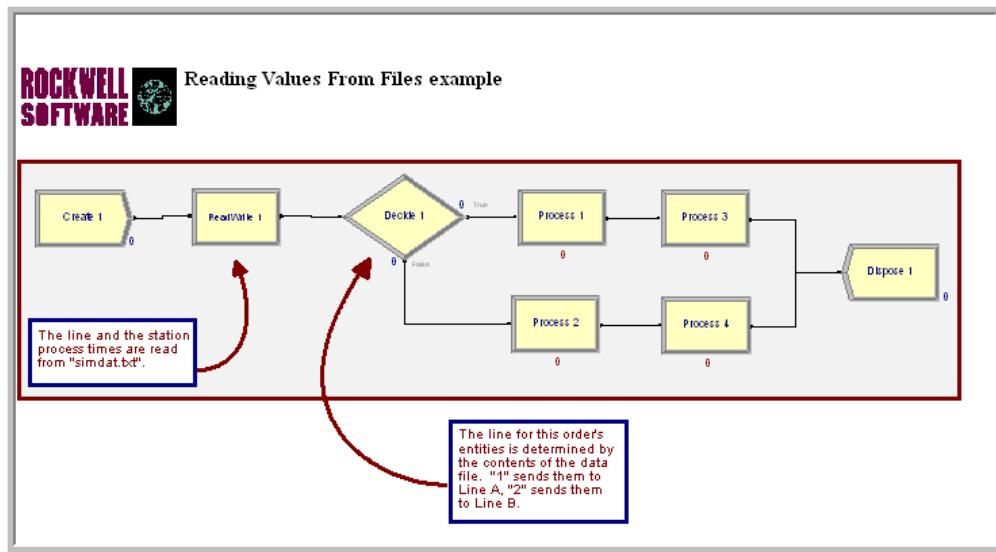


Figure D.26.: Smarts 162 Model

station 2 processing times, as shown in Figure D.27. In Figure D.28, the READWRITE module reads directly from the SIMDAT file using a free format. Each 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.

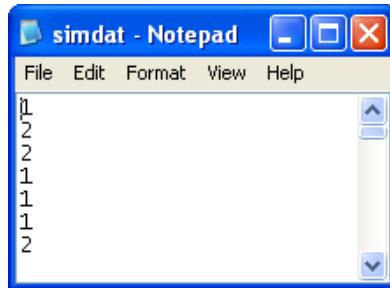


Figure D.27.: Sample processing times is simdat.txt

In Figure D.29, 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. 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
```

The *Initialize* option indicates what 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.

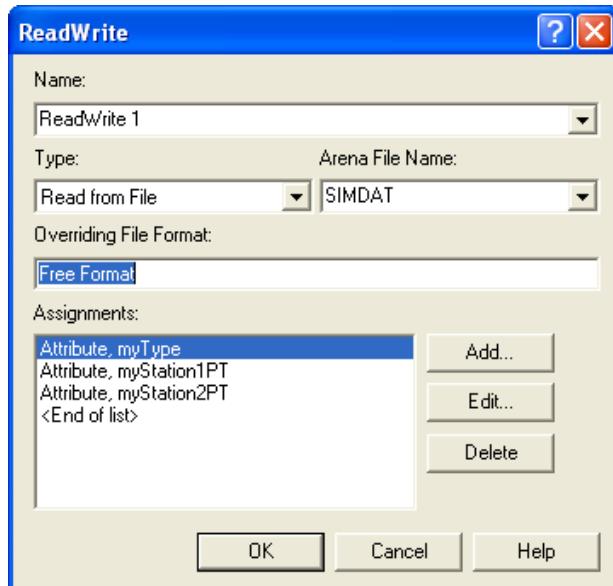


Figure D.28.: READWRITE module for simdat.txt

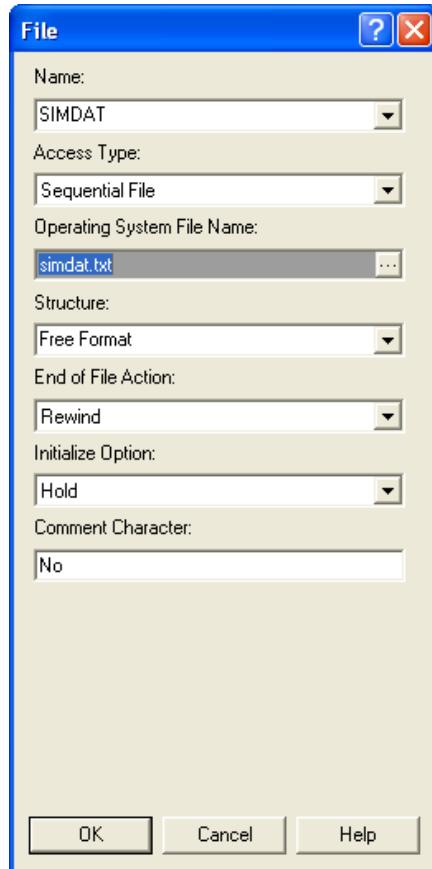


Figure D.29.: FILE module for Smarts162Revised.doe

D.3.2.2. Reading a Two Dimensional Array

Smarts file, *Smarts164.doe*, Figure D.30, shows how to read into a two-dimensional array. 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 4.

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 D.31. 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.

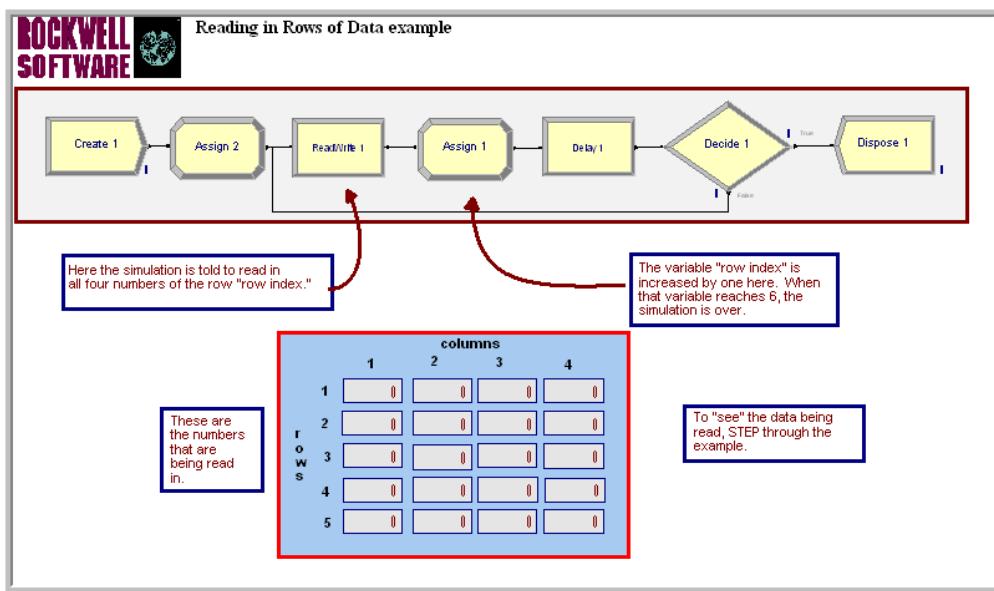


Figure D.30.: Smarts164.doe reading in a 2-D array

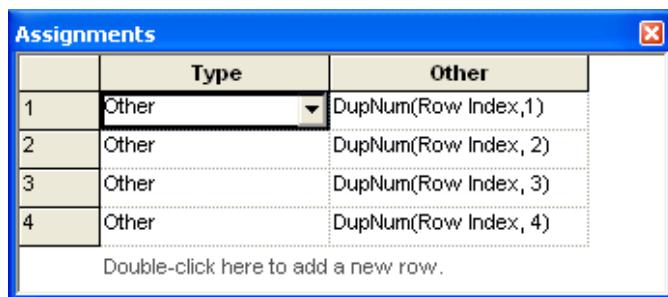


Figure D.31.: READWRITE assignments module using arrays

D.3.2.3. 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 the data to files, you can read from Excel files. Smarts file 185, Figure D.32, shows how to read in values from an Excel named range.

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 D.33. 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.

D. Miscellaneous Topics in Arena

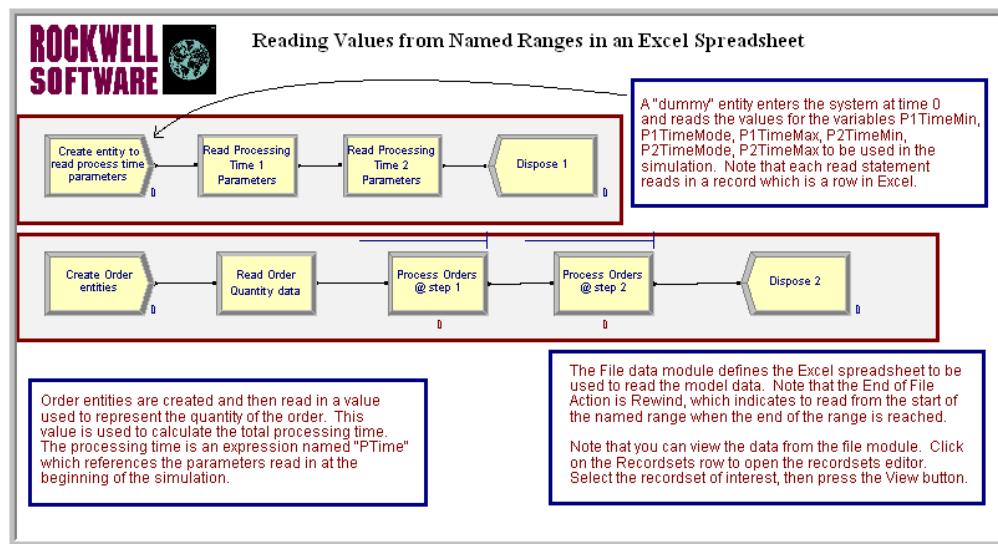


Figure D.32.: Smarts185.doe reading from an Excel named range

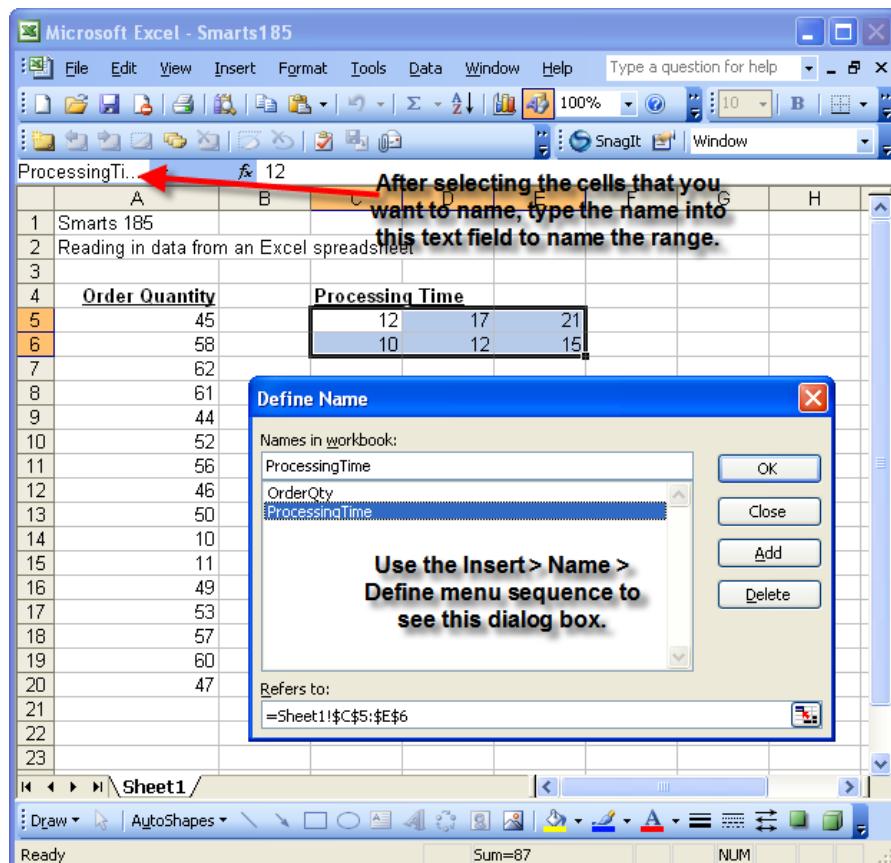


Figure D.33.: Checking the named range in Excel

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 D.32 there are two back to back READWRITE modules in the top level create logic to read in each of the rows associated with the processing times. In the bottom create logic, each new entity reads in a new row from the named range.

	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 D.34.: Data sheet view of FILE module

After setting up the spreadsheet and defining the named ranges within Excel, you must then 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, Figure D.34, 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. Then, click on the define Recordsets row button.

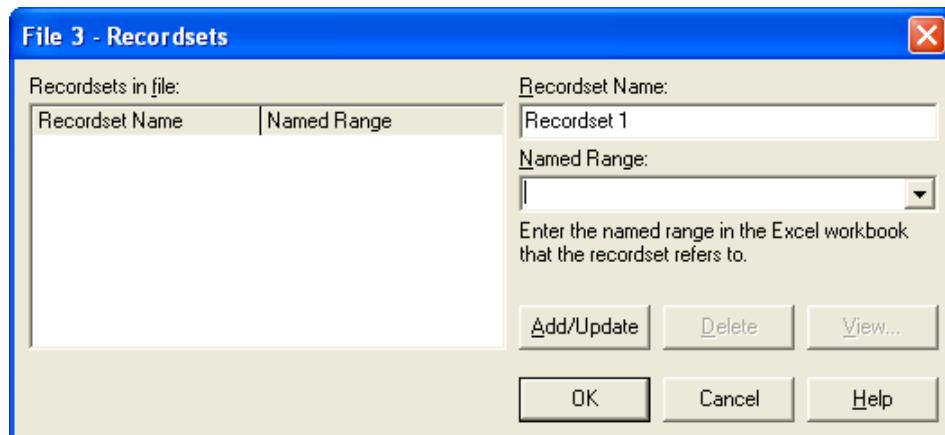


Figure D.35.: Defining the recordset for the FILE module

You will see a dialog box similar to Figure D.35. 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 D.36. 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

D. Miscellaneous Topics in Arena

a view of the data similar to that shown in Figure D.36. As you can see, it is very simple to define a named range and connect to it.

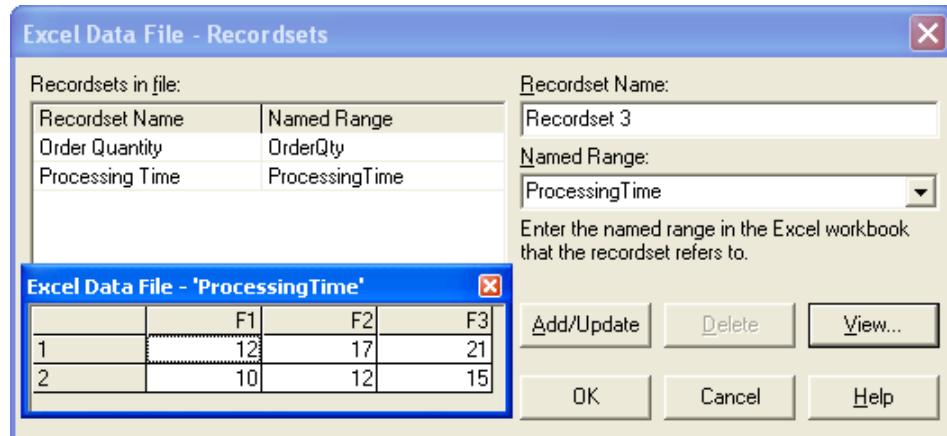


Figure D.36.: Viewing the ProcessingTime named range

Now, you have to indicate how to use the named range within the READWRITE module. Open up the READWRITE module, see Figure D.37, for reading the processing time 1 parameters. After selecting read from file and the associated file name, 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 Access 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:

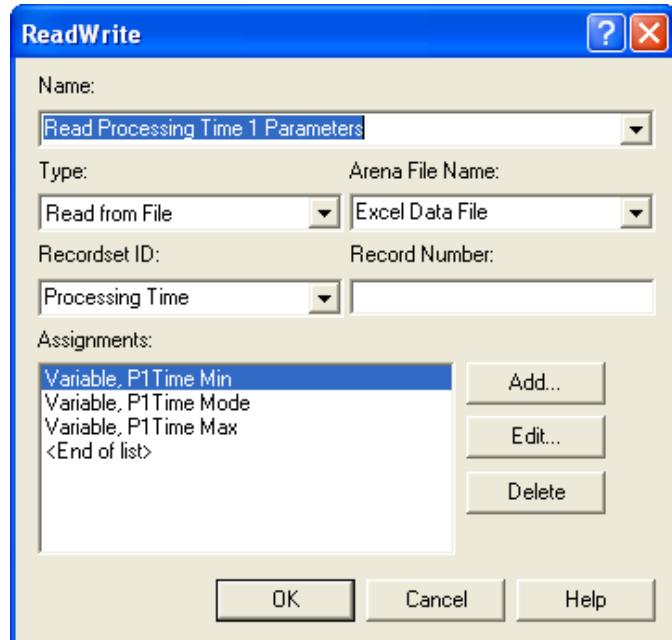


Figure D.37.: READWRITE module using named range

- Reading in the parameters of distributions
- Reading in model parameters (e.g. reorder points, order quantities, etc.)
- 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

D.3.2.4. 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 D.38 will not be discussed.

D. Miscellaneous Topics in Arena

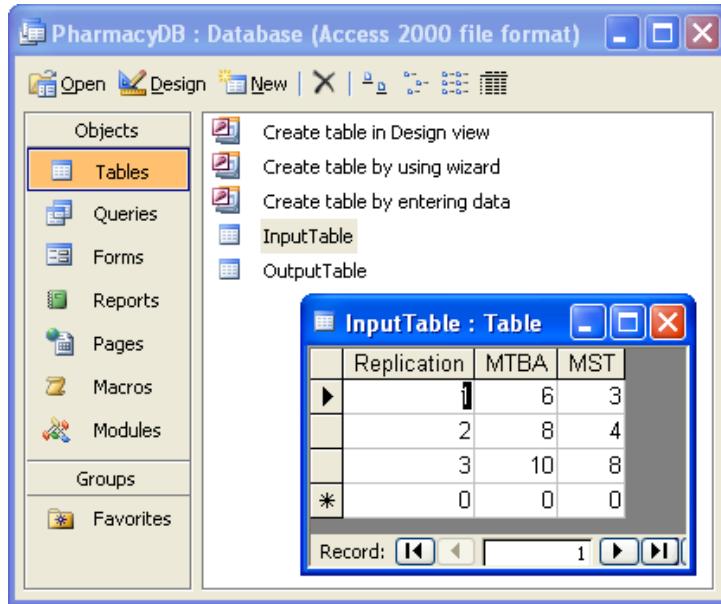


Figure D.38.: PharmacyDB Access Database

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 3 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 .

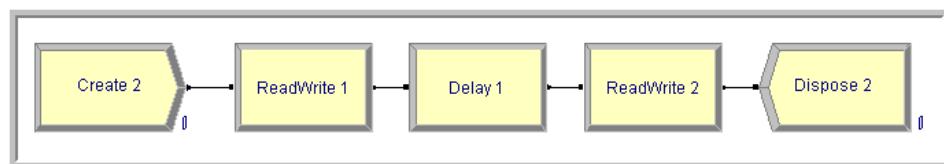


Figure D.39.: Read and write logic for Access example

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 D.39. An entity should be created at time zero, 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 D.39. Now, the 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 D.40 shows the basic setup in the data sheet view. Opening up the recordset rows allows you to define the tables as recordsets. You should define the recordsets as indicated in the figure.

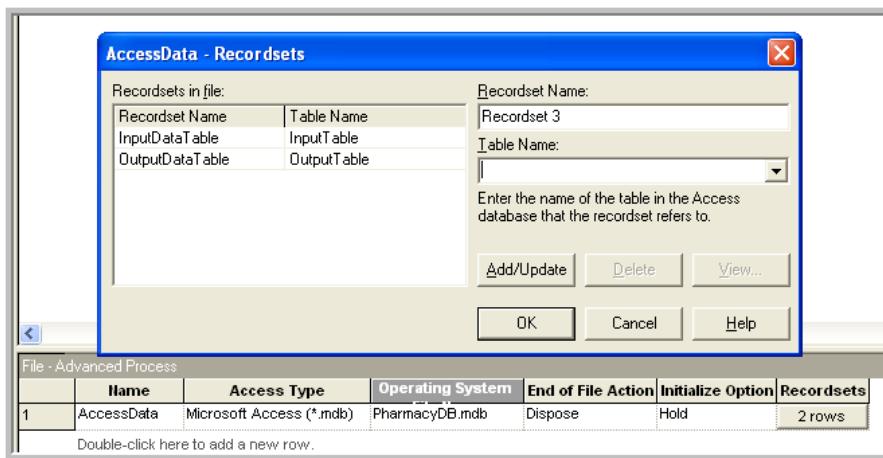


Figure D.40.: Read and write logic for Access example

The two READWRITE modules are quite simple. In the first READWRITE module (Figure D.41), the variables `RepNum`, `MTBA`, and `MST` are assigned values from the file. This will occur at time zero and thereby set the parameters for the run. The second READWRITE module (Figure D.42) writes out the value of the `RepNum` and uses the functions `TAVG()` and `THALF()` to get the values of the statistics associated with the waiting time in queue.

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 3 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 D.43.

Only one final module is left to edit. Open up the DELAY module. See Figure D.44. The entity entering this module should delay for the length of the simulation. has a special purpose variable called `TFIN` which holds the length of the simulation in *base time 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. You should see the results for each of the three replications.

D. Miscellaneous Topics in Arena

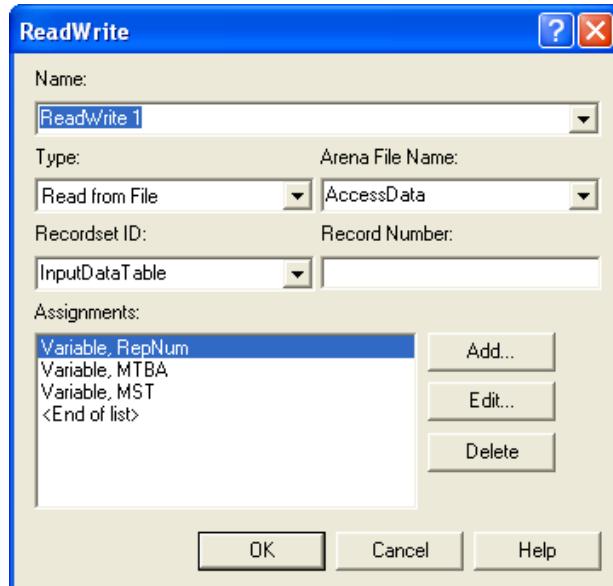


Figure D.41.: READWRITE assignments in first READWRITE module

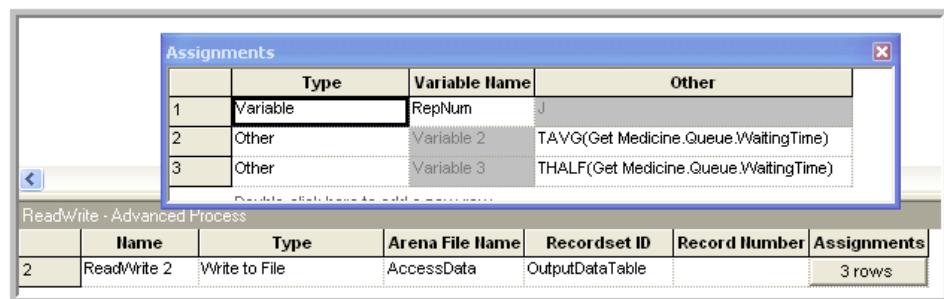


Figure D.42.: READWRITE assignments in second READWRITE module

Suppose now you wanted to replicate each run involving the parameter settings 3 times. All you would need to do would be to set up your Access input table as shown in Figure D.45 and change the number of replications to 9 in the Run Setup dialog.

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.

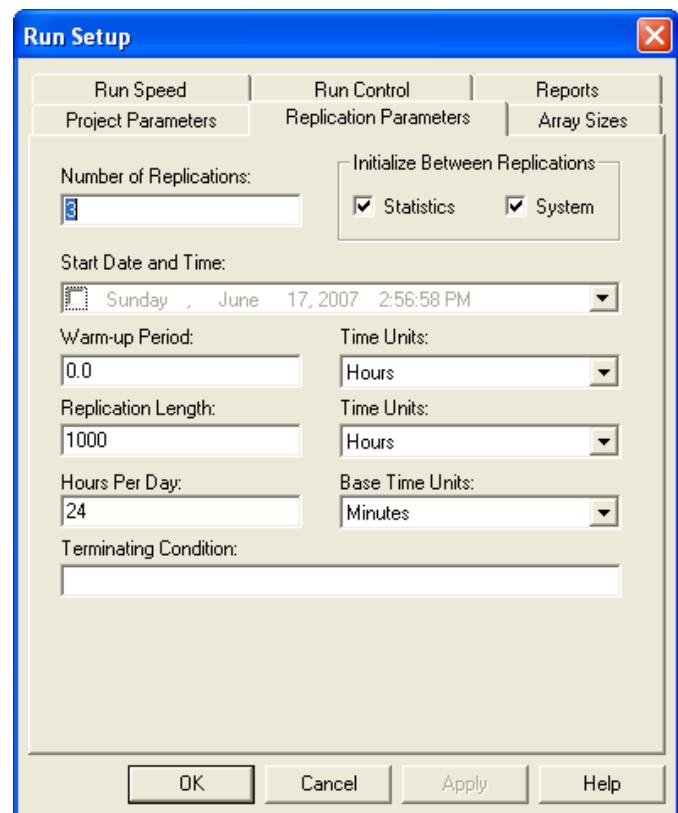


Figure D.43.: Run setup parameters for the database example

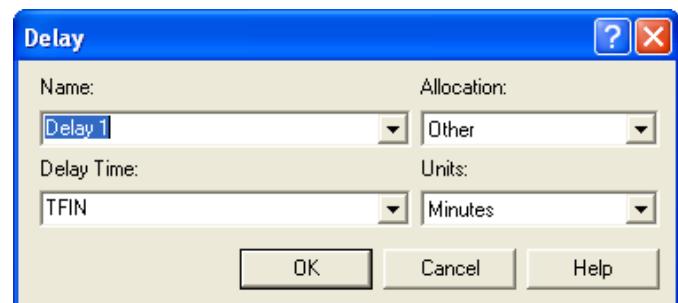


Figure D.44.: Delaying for the length of the replication

	Replication	AvgWaitingTime	HWWWaitingTime
▶	1	3.25356452762425	0.294470922917716
◀	2	3.81602561568035	0.373565865866371
▶	3	27.7529649795808	6.52722455471652
*	0	0	0

Record: [◀] [◀] [1] [▶] [▶] [▶] [▶] of 3

Figure D.45.: Output within the Access database

	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

Record: [◀] [◀] [10] [▶] [▶]

Figure D.46.: Making repeated replications

D.3.3. Using Visual Basic for Applications

This section discusses the relationship between and Visual Basic for Applications (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 often does not require the user to delve into VBA. The interaction between and VBA will be illustrated through a discussion of the VBA block and the use of Arena's user defined function. Through VBA, also has the ability to create models (e.g. lay down modules, fill dialogs, etc) 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 on-line help system.

To understand the connection between and VBA, you must understand the user event oriented nature of VBA. Do not confuse the user interaction events discussed in this section with dis-

crete events that occur within a simulation. Visual basic 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, clicks a button, etc.) 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 non-user 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. 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 .

DocumentOpen, DocumentSave These events are called when the model is opened or saved. The SIMAN object is not available, but the 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 object can be used. This event occurs automatically when the user invokes a simulation run. The 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 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.

D. Miscellaneous Topics in Arena

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 module.

D.3.3.1. Using VBA

Let us take a look at how to write a VBA event for responding to the *RunBegin* event. The file, *VBAEvents.doe* that is available with this chapter, should be opened. The 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 environment use the Tools > Macro > Show Visual Basic Editor menu option (as shown in Figure D.47).

This will open the VBA Editor as shown in Figure D.48. If you double click on the *ThisDocument* item in the VBA projects tree as illustrated in Figure D.48, you will open up a VBA module that is specifically associated with the current model.

A number of VBA events (Figure D.49) have already been defined for the model. Let's 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 VBA events that are available, as shown in Figure D.49. The event routines that have already been 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.

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

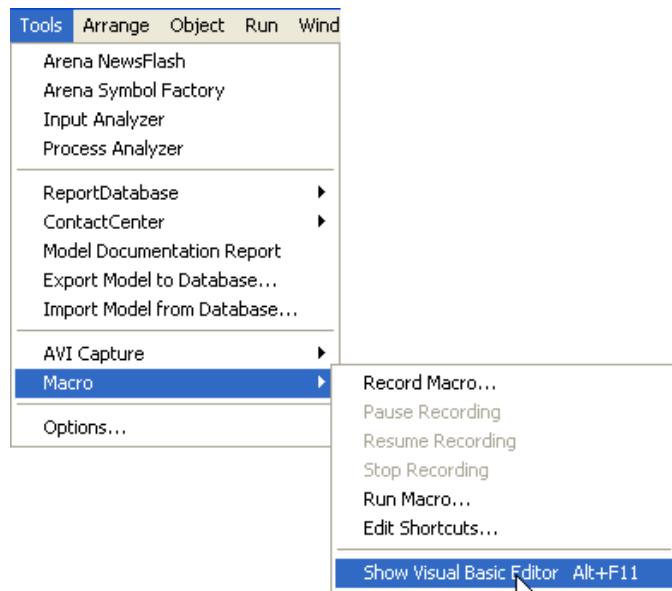


Figure D.47.: Showing the Visual Basic editor

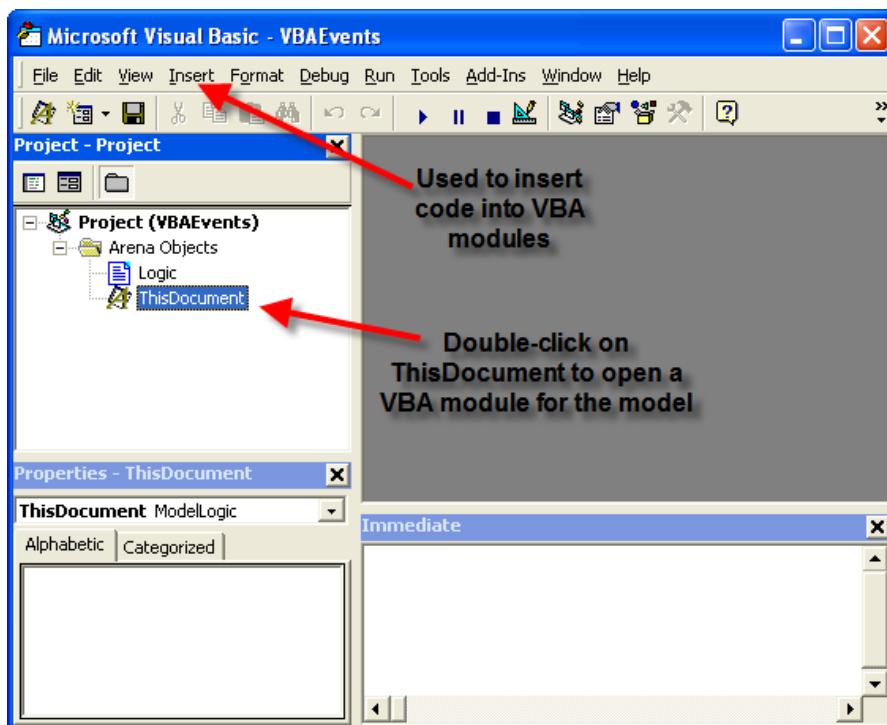


Figure D.48.: Showing the Visual Basic editor

D. Miscellaneous Topics in Arena

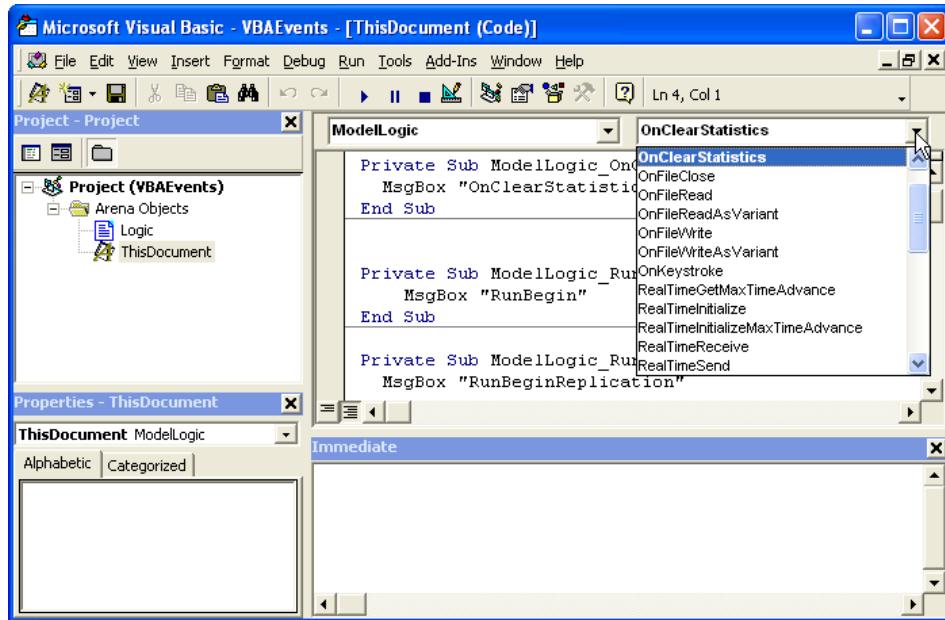


Figure D.49.: Showing the VBA Events for Arena

be very simple, it will just open up a message box that indicates that the *RunBegin* VBA event has occurred. The code to open up a message box when the event procedure fires is as follows:

```

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 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 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 Smart files are a good source of examples related to VBA. The following models are located in the Smarts folder within your main folder (e.g., Program Files > Rockwell Software > Smarts).

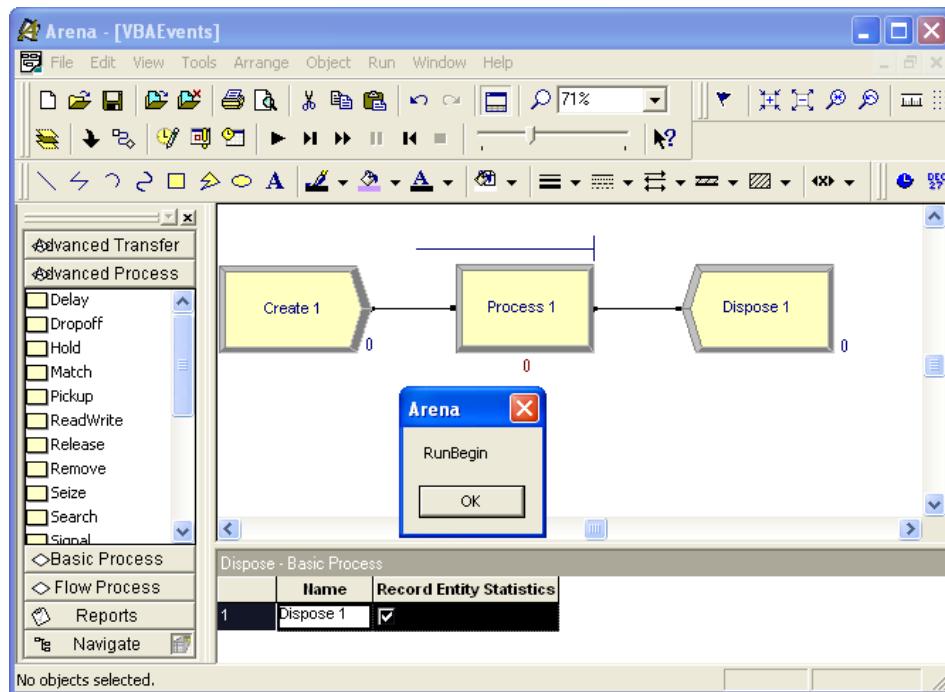


Figure D.50.: Message Box for RunBegin example

- 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

D. Miscellaneous Topics in Arena

- 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
- 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 section, the UserFunction event and the VBA block are illustrated.

D.3.3.2. The VBA Module and User Defined Functions

This section presents how to 1) use a user form to set the value of a variable, 2) call a user defined function which uses the value of an attribute to compute a quantity, and 3) 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 D.51. 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 D.51. 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.

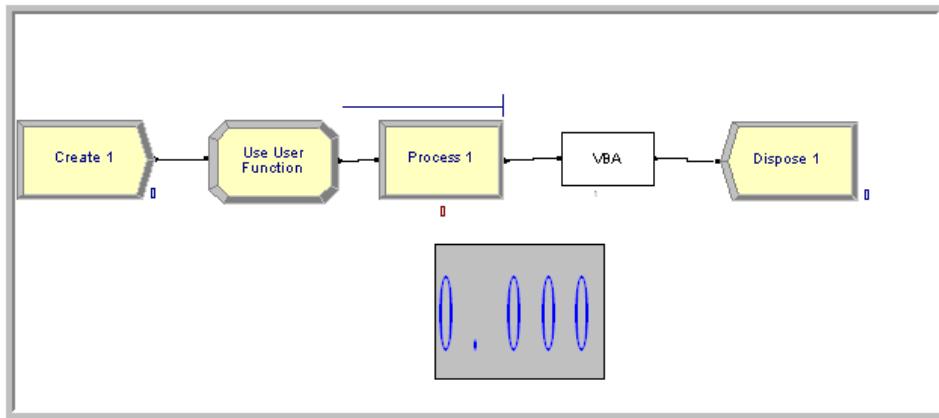


Figure D.51.: Simple VBA example model

- 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 completing these steps you can look at the module dialogues in the file called `VBAExample.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 the following code. If you don't want to retype this code, then you can access it in the file `VBAExample.doe`.

```

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

```

D. Miscellaneous Topics in Arena

```

' 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 help system by searching on *Automation Programmer's Reference*. The SIMAN object is created after the simulation has been compiled and essentially gives access to the underlying simulation engine. The variables `vPTFactorIndex` and `myPTIndex` will be used to index into SIMAN to access the 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 D.52.

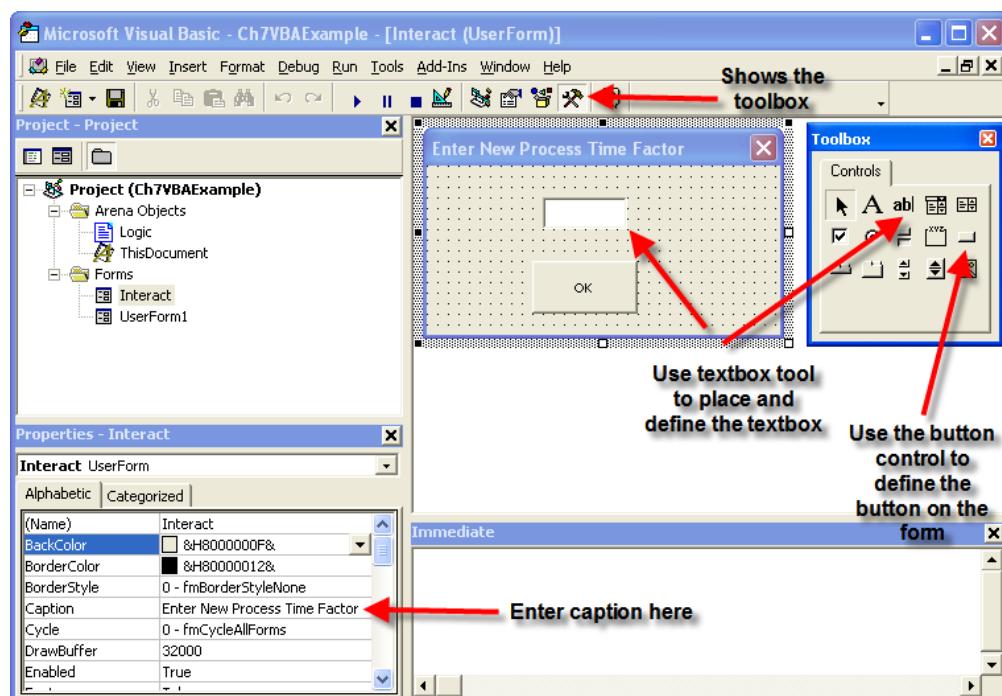


Figure D.52.: Building the Interact form

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 Figure D.52 and Figure D.53. To place the labels used in the forms,

use the label control (right next to the textbox control on the Toolbox). The name of a form can be changed in the Misc > (Name) property as shown in Figure D.54. Now that the forms have been built, the controls on the forms can be referenced within other VBA modules.

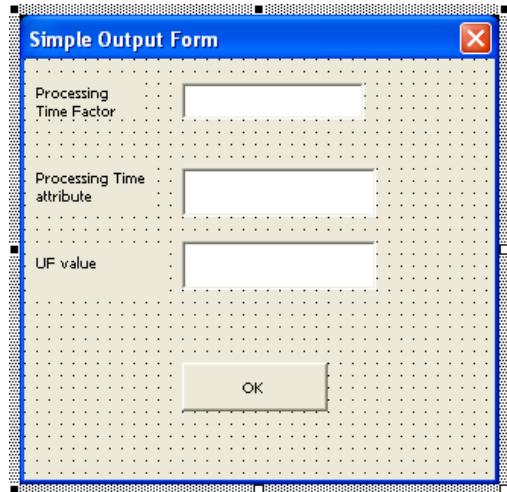


Figure D.53.: VBA UserForm1

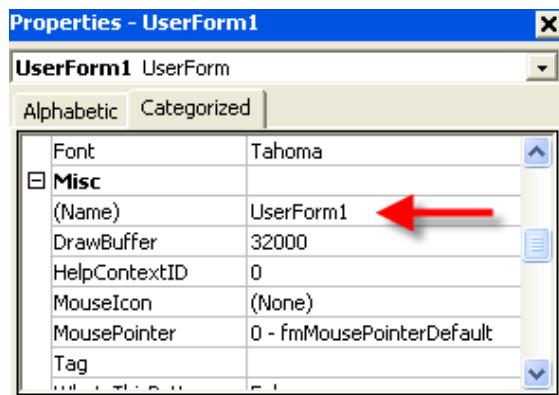


Figure D.54.: Properties Window

Before looking at the code for this situation, we need to understand how to interchange data between the 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 help system on SIMAN Object (Automation). There are many properties and methods associated with the SIMAN Object. The documentation states the following concerning the `SymbolNumber` method.

SymbolNumber Method

D. Miscellaneous Topics in Arena

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`, etc. 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 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. The VBA code shows 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 exhibit, 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.

```
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 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 next shows the use of the `VariableArrayValue` property to assign the value entered into the textbox 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.

```

Private Sub CommandButton2_Click()
' when the user clicks the ok button
' Set the current value of vPT to the value

```

D. Miscellaneous Topics in Arena

```

    ' 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 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 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. The following code exhibit shows the code for the UF function for this example.

```

    ' 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

```

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 similar fashion. 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 within the model by supplying the appropriate function identifier.

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 one. 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.*

D. Miscellaneous Topics in Arena

Let's see how to put these methods into action within the user defined functions. The following code exhibit shows how to access the value of an attribute associated with the active entity.

```
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
```

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.

Within Arena, attributes can be defined as multi-dimensional via the ATTRIBUTES module. Multi-dimensional 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 multi-dimensional 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 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 model. The *uf* function is especially useful for returning a value back to the model. 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 model, each VBA block is given a unique number. Then, within the *ThisDocument* module, an individual event routine can be created that is associated

with each VBA block. 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 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. The following exhibit shows the code for the 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 *textboxes* 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 the next exhibit.

```
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 don't 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.

D.3.3.3. 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

D. Miscellaneous Topics in Arena

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 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 Transformation as discussed in Banks et al. (2005), (Cario and Nelson, 1998), (Cario and Nelson, 1996), and (Biller and Nelson, 2003). Suppose you have a $N(0, 1)$ random variable, Z_i , and a way to compute the 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.

$$X_i = F^{-1}(U_i) = F^{-1}((\Phi(Z_i)))$$

This transform is called the normal-to-anything (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 ρ_z then the X_i will have ρ_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, i.e. AR(1). An AR(1) model with $N(0, 1)$ marginal distributions has the following form:

$$Z_i = \phi Z_{i-1} + \varepsilon_i$$

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:

$$\phi = \rho^1 = \text{corr}(Z_i, Z_{i+1})$$

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 ρ_z . By changing ϕ , 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 NOR-TAExample.xls can also be used to perform this search process.

The implementation of this technique cannot be readily achieved in a general way within 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 the following code exhibit.

```
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.

The following code exhibit 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.

```
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
```

D. Miscellaneous Topics in Arena

```
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 ϕ 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 does not have built in (e.g. binomial).

Arena 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 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 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 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. 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 . For more information on this topic, you should search the help system under *Introduction to C Support*.

D.4. 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. Activity based costing 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 activity based costing will not be discussed within this text; however, how tabulates costs will be examined so that you can understand how to utilize the cost estimates available on the summary reports.

D.4.1. 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's 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 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/hour and the mailing worker is paid \$5.15/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, 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 D.55.

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 D.55.: Specifying resource costs

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. 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 D.56.

D. Miscellaneous Topics in Arena

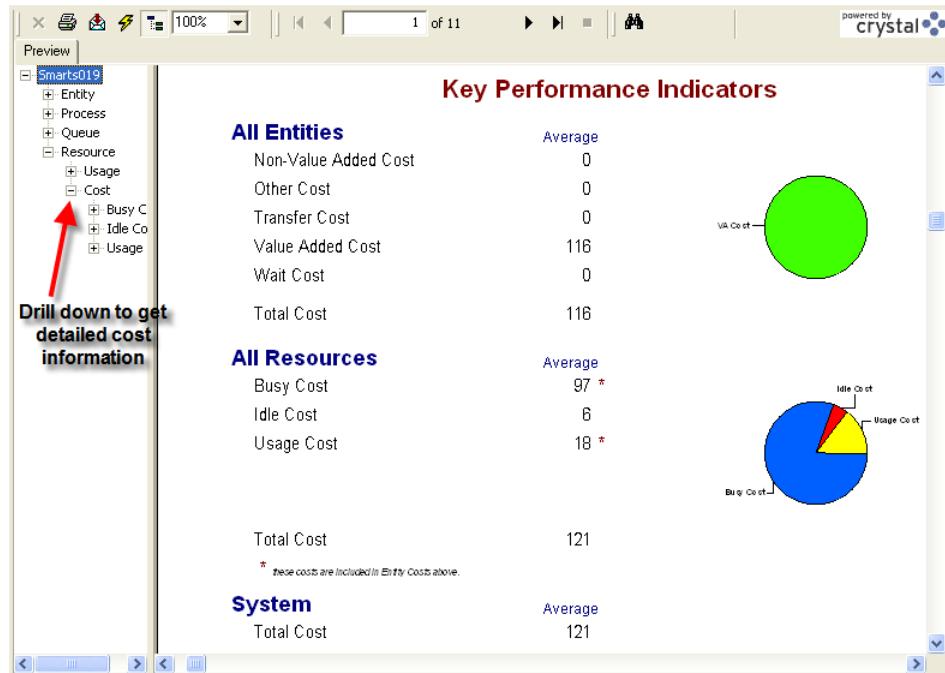


Figure D.56.: Resource cost summary

Table D.1 and Table D.2 indicate how the costs are tabulated from Arena's statistical reports for the resources. In Table D.1, the number of uses is known because there were 435 entities (bills) processed by the system. In Table D.2 the utilization from the resource statistics was used to estimate the percentage of time busy and idle. From this, the amount of time of time can be calculated and from that the cost. tabulates the actual amount of time in these categories. The value added cost calculation will be discussed shortly.

Table D.1.: Tabulation of resource costs

Resource cost	\$/Hour	Hours	Cost	Totals
Biller	7.75	8	62.0	
Mailer	5.15	8	41.2	
				103.2
Usage Cost	\$/Usage	# Uses	Cost	
Biller	0.02	456	9.12	
Mailer	0.02	456	9.12	
				18.24
		Total Cost	121.44	

Table D.2.: Tabulation of busy and idle costs

Resource	Util.	Time Busy	\$/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	\$/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		

To contrast this consider *Smarts049.doe* in which the billing worker and mailing workers follow a schedule as shown in Figure D.57 and Figure D.58. Because the resources have less time available to be busy or idle the costs are less for this model as shown in Figure D.59. In this case, the workers are not paid when they are not scheduled. As can be seen in Figure D.59, Arena also tabulates costs for the entities. Let's take a look at another example to examine how entity costs are tabulated.

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	VWait	5.15	5.15	.02
Double-click here to add a new row.							

Figure D.57.: Smart049.doe with resources based on schedule

D. Miscellaneous Topics in Arena

Schedule - Basic Process

	Name	Format Type	Type	Time Units	Scale Factor	Durations
1	Schedule 1	Duration	Capacity	Minutes	1.0	8 rows

Double-click here to add a new row.

Durations

	Value	Duration
1	1	135
2	0	15
3	1	90
4	0	60
5	1	90
6	0	15
7	1	45
8	1	Infinite

Double-click here to add a new row.

Figure D.58.: Schedule for Smarts049.doe

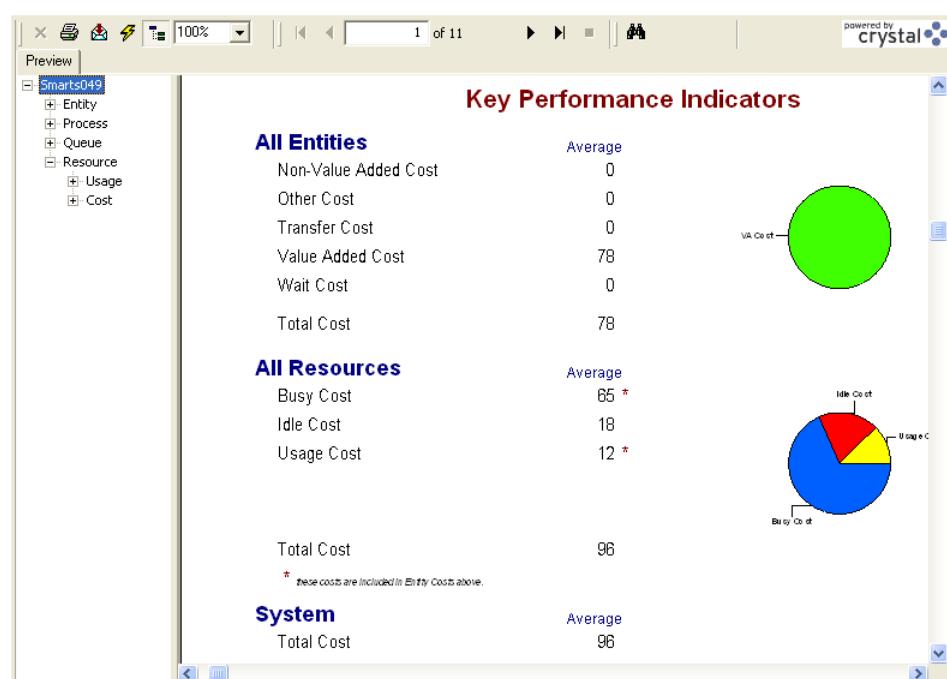


Figure D.59.: Summary costs for resources following a schedule

D.4.2. Entity Costing

Entity costing is slightly more complex than resource costing. 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.

Non-value added time/cost Non-value added time is any time designated as Non-Value added. Non-value 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 non-value added. The designation of non-value 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 non-value added. Non-value added cost is the cost associated with an entity's designated non-value added time. This cost includes the value of the time the entity spends in non-value 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 non-value 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, 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, does not do the cost modeling for you; however, it tabulates the costs automatically for you. Don't forget the golden rule of computing: "Garbage in = Garbage out". When you want to use for cost modeling, you need to *carefully* specify the cost elements within the *entire* model. Let's take a closer look at how tabulates the costs.

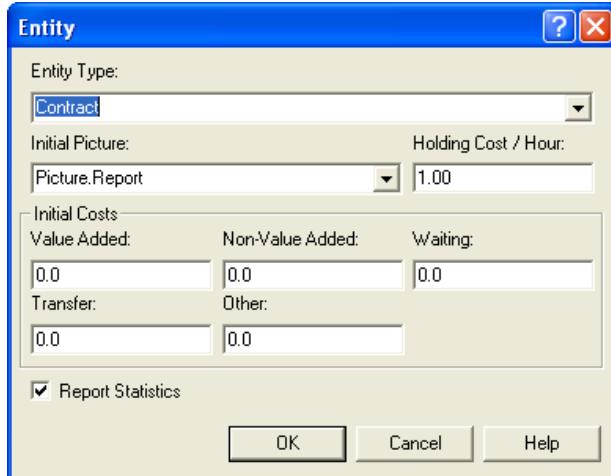


Figure D.60.: ENTITY Module Costing Terms for Smarts047.doe

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 D.60, 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's ignore the initial costs for a moment and examine the meaning of the holding cost/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, can tabulate the costs.

Let's 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:

$$TVAC = \sum_{i=1}^n VAC_i$$

The costs for the other categories are computed in a similar fashion 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/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 Figure D.61, Figure D.60, and Figure D.62 illustrates how to allocate the value added time within the contract addendum process.

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 D.61.: Resource Costs for Smarts047.doe

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 D.63 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 D.3. 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

D. Miscellaneous Topics in Arena

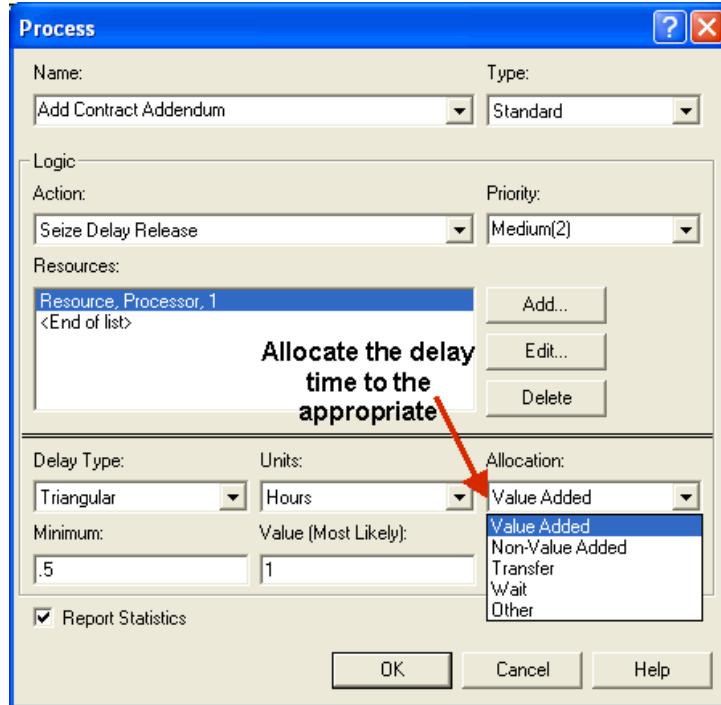


Figure D.62.: Allocating the value added time

resource for the process has a resource cost then it must be included as shown in Table D.3 and as explained in the equation for VAC_i .

Table D.3.: Tabulation of entity cost

Process	VAT (minutes)	VAT (hours)	Holding Cost(\$/hr)	Resource Cost(\$/hr)	Cost(\$/hr)	Usage Cost	Total Cost
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

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 D.3.

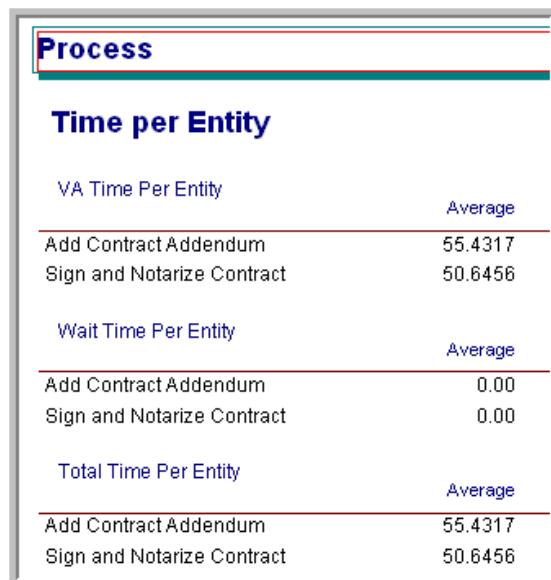


Figure D.63.: Processing time

- 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 module's 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.
- Read carefully how handles PROCESS sub-models and regular sub-models in terms of costing tabulation if you use sub-models. See *Sub-model 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.

D.5. Summary

As you might note from the hodge-podge of topics in this appendix, there are many technical details of using Arena. This appendix (and this book) in general, can be used as a resource to get you started, but it should not be considered an exhaustive representation of the technical details of Arena. For example, Arena allows modelers to develop modeling templates and also has other advanced templates for modeling packaging, tanks and piping, etc. that are not even mentioned within this book. For additional details on some of these topics, I recommend using the Arena help system as well as the textbook by (Kelton et al., 2015).

E. Arena Operators, Functions, Distributions, and Modules

E.1. Arena Mathematical and Logical Operators

Table E.1.: Mathematical and Logical Operators

Operator	Operation	Priority
Math Operators		
**	Exponentiation	1(highest)
/	Division	2
*	Multiplication	2
-	Subtraction	3
+	Addition	3
Logical Operators		
.EQ., ==	Equality comparison	4
.NE., <>	Non-equality 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 E.2.: Arena's Mathematical Functions

Function	Description
ABS(a)	Absolute value
ACOS(a)	Arc cosine
AINT(a)	Truncate
AMOD(a1, a2)	Real remainder, returns($a1 - (AINT(\frac{a1}{a2}) \times a2)$)
ANINT(a)	Round to nearest integer
ASIN(a)	Arc sine
ATAN(a)	Arc tangent
COS(a)	Cosine
EP(a)	Exponential(e^a)
HCOS(a)	Hyperbolic cosine
HSIN(a)	Hyperbolic sine
HTAN(a)	Hyperbolic tangent
MN(a1, a2, ...)	Minimum value
MOD(a1,a2)	same as AMOD(AINT(a1), AINT(a2))
MX(a1, a2, ...)	Maximum value
LN(a)	Natural logarithm
LOG(a)	Common logarithm
SIN(a)	Sine
SQRT(a)	Square root
TAN(a)	Tangent

E.2. Arena Probability Distributions Functions

Table E.3.: Arena's Distributions

Beta	BETA(Alpha, Alpha2[, stream])
Normal	NORM(Mean,SD[, Stream])
Empirical Continuous	CONT(Prob1,Value1,Prob2,Value2,...[, Stream])
NSExpo	Non-homogeneous 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])

E.3. Basic Process Panel Modules

Table E.4.: 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.

E.4. Advanced Process Panel Modules

Table E.5.: 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.

E.5. Advanced Transfer Panel Modules

Table E.6.: 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 entity a transporter without moving the transporter, entity controls transporter and may wait in queue.
MOVE	Moves an allocated transporter to a station destination.
REQUEST	Asks transporter for pick up, entity waits until transporter is allocated, and transporter moves to pick up 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 pre-specified routes of stations to be defined and attributes to be assigned when entities are transferred.
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 for the collection of aggregate statistics on the group of stations.

E.6. Important SIMAN Blocks, Elements, and Pre-Defined Attributes and Variables

Table E.7.: Miscellaneous Useful Blocks, Attributes, and Variables

IF-ELSEIF-ELSE-ENDIF	(Blocks panel) Allows standard logic based flow of control.
WHILE-ENDWHILE	(Blocks panel) Allows for iterative looping.
BRANCH	(Blocks panel) Allows probabilistic and condition based path determination and 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.

F. Distributions

F.1. Discrete Distrbutions

Bernoulli	$Ber(p)$
Parameters:	$0 < p < 1$, probability of success
PMF:	$P[X = 1] = p, P[X = 0] = 1 - p$
Inverse CDF:	$F^{-1}(u) = \text{if } (u < p), 1 \text{ else } 0$
Expected Value:	$E[X] = p$
Variance:	$Var[X] = p(1 - p)$
Arena Generation:	<code>DISC(p,1,1.0,0[,stream])</code>
Spreadsheet Generation:	<code>= IF(RAND0 < p, 1, 0)</code>
Modeling:	the number of successes in one trial

Binomial	$Binom(n, p)$
Parameters:	$0 < p < 1$, probability of success, n , number of trials
PMF:	$P[X = x] = \binom{n}{x} p^x (1 - p)^{n-x} \quad x = 0, 1, \dots, n$
Inverse CDF:	no closed form available
Expected Value:	$E[X] = np$
Variance:	$Var[X] = np(1 - p)$
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

Shifted Geometric	Shifted Geo(p)
Parameters:	$0 < p < 1$, probability of success
PMF:	$P[X = x] = p(1 - p)^{x-1} \quad x = 1, 2, \dots,$
Inverse CDF	$F^{-1}(u) = 1 + \left\lfloor \frac{\ln(1-u)}{\ln(1-p)} \right\rfloor$
Expected Value:	$E[X] = 1/p$
Variance:	$Var[X] = (1 - p)/p^2$
Arena Generation:	<code>1 + AINT(LN(1-UNIF(0,1))/LN(1-p))</code>
Spreadsheet Generation:	<code>= 1 + INT(LN(1-RAND())/LN(1-p))</code>
Modeling:	the number of trials until the first success

F. Distributions

Negative Binomial Defn. 1 NB1(r, p)	
Parameters:	$0 < p < 1$, probability of success, r^{th} success
PMF:	$P[X = x] = \binom{x-1}{r-1} p^r (1-p)^{x-r} \quad x = r, r+1, \dots$
Inverse CDF:	no closed form available
Expected Value:	$E[X] = r/p$
Variance:	$Var[X] = r(1-p)/p^2$
Arena Generation:	use convolution of shifted geometric
Spreadsheet Generation:	use convolution of shifted geometric
Modeling:	the number of trials until the r^{th} success

Negative Binomial Defn. 2 NB2(r, p)	
Parameters:	$0 < p < 1$, probability of success, r^{th} success
PMF:	$P[Y = y] = \binom{y+r-1}{r-1} p^r (1-p)^y \quad y = 0, 1, \dots$
Inverse CDF:	no closed form available
Expected Value:	$E[Y] = r(1-p)/p$
Variance:	$Var[Y] = r(1-p)/p^2$
Arena Generation:	use convolution of geometric
Spreadsheet Generation:	use convolution of geometric
Modeling:	the number of failures prior to the r^{th} success

Poisson	Pois(λ)
Parameters:	$\lambda > 0$
PMF:	$P[X = x] = \frac{e^{-\lambda} \lambda^x}{x!} \quad x = 0, 1, \dots$
Inverse CDF:	no closed form available
Expected Value:	$E[X] = \lambda$
Variance:	$Var[X] = \lambda$
Arena Generation:	POIS(λ [, stream])
Spreadsheet Generation:	not available, approximate with lookup table approach
Modeling:	the number of occurrences during a period of time

Discrete Uniform	$DU(a, b)$
Parameters:	$a \leq b$
PMF:	$P[X = x] = \frac{1}{b-a+1} \quad x = a, a+1, \dots, b$
Inverse CDF:	$F^{-1}(u) = a + \lfloor (b-a+1)u \rfloor$
Expected Value:	$E[X] = (b+a)/2$
Variance:	$Var[X] = ((b-a+1)^2 - 1)/12$
Arena Generation:	$a + AINT((b-a+1)^*UNIF(0,1))$
Spreadsheet Generation:	=RANDBETWEEN(a,b)
Modeling:	equal occurrence over a range of integers

F.2. Continuous Distrbutions

Uniform	$U(a, b)$
Parameters:	$a = \text{minimum}, b = \text{maximum}, -\infty < a < b < \infty$
PDF:	$f(x) = \frac{1}{b-a}$ for $a \leq x \leq b$
CDF:	$F(x) = \frac{x-a}{b-a}$ if $a \leq x \leq b$
Inverse CDF:	$F^{-1}(p) = a + p(b-a)$ if $0 < p < 1$
Expected Value:	$E[X] = \frac{a+b}{2}$
Variance:	$Var[X] = \frac{(b-a)^2}{12}$
Arena Generation:	$UNIF(a,b[,Stream])$
Spreadsheet Generation:	=a + RAND()*(b-a)
Modeling:	assumes equally likely across the range, when you have lack of data, task times

Normal	$N(\mu, \sigma^2)$
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:	$NORM(\mu, \sigma^2 [,Stream])$
Spreadsheet Generation:	=NORM.INV(RAND(), μ , σ)
Modeling:	task times, errors

F. Distributions

Exponential	EXPO(1/ λ)
Parameters:	$\lambda > 0$
PDF:	$f(x) = \lambda e^{-\lambda x}$ if $x \geq 0$
CDF:	$F(x) = 1 - e^{-\lambda x}$ if $x \geq 0$
Inverse CDF:	$F^{-1}(p) = (-1/\lambda) \ln(1-p)$ if $0 < p < 1$
Expected Value:	$E[X] = \theta = 1/\lambda$
Variance:	$Var[X] = 1/\lambda^2$
Arena Generation:	EXPO($\theta[, \text{stream}]$)
Spreadsheet Generation:	= (-1/ λ)LN(1-RAND())
Modeling:	time between arrivals, time to failure highly variable task time

Weibull	WEIB(β, α)
Parameters:	$\beta > 0$ (scale), $\alpha > 0$ (shape)
CDF:	$F(x) = 1 - e^{-(x/\beta)^\alpha}$ if $x \geq 0$
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:	$Var[X] = \left(\frac{\beta^2}{\alpha}\right) \left\{ 2\Gamma\left(\frac{2}{\alpha}\right) - \left(\frac{1}{\alpha}\right) \left(\Gamma\left(\frac{1}{\alpha}\right)\right)^2 \right\}$
Arena Generation:	WEIB(scale, shape[, stream])
Spreadsheet Generation:	= (β)(-LN(1 - RAND()) \wedge (1/ α))
Modeling:	task times, time to failure

Erlang	Erlang(r, β)
Parameters:	$r > 0$, integer, $\beta > 0$ (scale)
CDF:	$F(x) = 1 - e^{(-x/\beta)} \sum_{j=0}^{r-1} \frac{(x/\beta)^j}{j!}$ if $x \geq 0$
Inverse CDF:	No closed form
Expected Value:	$E[X] = r\beta$
Variance:	$Var[X] = r\beta^2$
Arena Generation:	ERLA($E[X], r[, \text{stream}]$)
Spreadsheet Generation:	= GAMMA.INV(RAND(), r, β)
Modeling:	task times, lead time, time to failure,

Gamma	Gamma(α, β)
Parameters:	$\alpha > 0$, shape, $\beta > 0$ (scale)
CDF:	No closed form

Gamma	Gamma(α, β)
Inverse CDF:	No closed form
Expected Value:	$E[X] = \alpha\beta$
Variance:	$Var[X] = \alpha\beta^2$
Arena Generation:	GAMM(scale, shape[, stream])
Spreadsheet Generation:	= GAMMA.INV(RAND(), α, β)
Modeling:	task times, lead time, time to failure,

Beta	BETA(α_1, α_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(α_1, α_2 [, stream])
Spreadsheet Generation:	BETA.INV(RAND(), α_1, α_2)
Modeling:	activity time when data is limited, probabilities

Lognormal	LOGN(μ_l, σ_l)
Parameters:	$\mu = \ln(\mu_l^2 / \sqrt{\sigma_l^2 + \mu_l^2})$ $\sigma^2 = \ln((\sigma_l^2 / \mu_l^2) + 1)$
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(μ_l, σ_l [, stream])
Spreadsheet Generation:	LOGNORM.INV(RAND(), μ, σ)
Modeling:	task times, time to failure

Triangular	TRIA(a, m, b)
Parameters:	a = minimum, m = mode, b = maximum
CDF:	$F(x) = \frac{(x - a)^2}{(b - a)(m - a)}$ for $a \leq x \leq m$ $F(x) = 1 - \frac{(b - x)^2}{(b - a)(b - m)}$ for $m < x \leq b$
Inverse CDF:	$F^{-1}(u) = a + \sqrt{(b - a)(m - a)u}$ for $0 < u < \frac{m - a}{b - a}$ $F^{-1}(u) = b - \sqrt{(b - a)(b - m)(1 - u)}$ for $\frac{m - a}{b - a} \leq u$

F. Distributions

Triangular	TRIA(a, m, b)
Expected Value:	$E[X] = (a + m + b)/3$
Variance:	$Var[X] = \frac{a^2 + b^2 + m^2 - ab - am - bm}{18}$
Arena Generation:	TRIA(min, mode, max[, stream])
Spreadsheet Generation:	implement $F^{-1}(u)$ as VBA function
Modeling:	task times, activity time when data is limited

G. Statistical Tables

G. Statistical Tables

Table G.1.: Cumulative standard normal distribution, z = -3.49 to 0

	-0.09	-0.08	-0.07	-0.06	-0.05	-0.04	-0.03	-0.02	-0.01	0
-3.40	0.0002	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003
-3.30	0.0003	0.0004	0.0004	0.0004	0.0004	0.0004	0.0004	0.0005	0.0005	0.0005
-3.20	0.0005	0.0005	0.0005	0.0006	0.0006	0.0006	0.0006	0.0006	0.0007	0.0007
-3.10	0.0007	0.0007	0.0008	0.0008	0.0008	0.0008	0.0009	0.0009	0.0009	0.0010
-3.00	0.0010	0.0010	0.0011	0.0011	0.0011	0.0012	0.0012	0.0013	0.0013	0.0013
-2.90	0.0014	0.0014	0.0015	0.0015	0.0016	0.0016	0.0017	0.0018	0.0018	0.0019
-2.80	0.0019	0.0020	0.0021	0.0021	0.0022	0.0023	0.0023	0.0024	0.0025	0.0026
-2.70	0.0026	0.0027	0.0028	0.0029	0.0030	0.0031	0.0032	0.0033	0.0034	0.0035
-2.60	0.0036	0.0037	0.0038	0.0039	0.0040	0.0041	0.0043	0.0044	0.0045	0.0047
-2.50	0.0048	0.0049	0.0051	0.0052	0.0054	0.0055	0.0057	0.0059	0.0060	0.0062
-2.40	0.0064	0.0066	0.0068	0.0069	0.0071	0.0073	0.0075	0.0078	0.0080	0.0082
-2.30	0.0084	0.0087	0.0089	0.0091	0.0094	0.0096	0.0099	0.0102	0.0104	0.0107
-2.20	0.0110	0.0113	0.0116	0.0119	0.0122	0.0125	0.0129	0.0132	0.0136	0.0139
-2.10	0.0143	0.0146	0.0150	0.0154	0.0158	0.0162	0.0166	0.0170	0.0174	0.0179
-2.00	0.0183	0.0188	0.0192	0.0197	0.0202	0.0207	0.0212	0.0217	0.0222	0.0228
-1.90	0.0233	0.0239	0.0244	0.0250	0.0256	0.0262	0.0268	0.0274	0.0281	0.0287
-1.80	0.0294	0.0301	0.0307	0.0314	0.0322	0.0329	0.0336	0.0344	0.0351	0.0359
-1.70	0.0367	0.0375	0.0384	0.0392	0.0401	0.0409	0.0418	0.0427	0.0436	0.0446
-1.60	0.0455	0.0465	0.0475	0.0485	0.0495	0.0505	0.0516	0.0526	0.0537	0.0548
-1.50	0.0559	0.0571	0.0582	0.0594	0.0606	0.0618	0.0630	0.0643	0.0655	0.0668
-1.40	0.0681	0.0694	0.0708	0.0721	0.0735	0.0749	0.0764	0.0778	0.0793	0.0808
-1.30	0.0823	0.0838	0.0853	0.0869	0.0885	0.0901	0.0918	0.0934	0.0951	0.0968
-1.20	0.0985	0.1003	0.1020	0.1038	0.1056	0.1075	0.1093	0.1112	0.1131	0.1151
-1.10	0.1170	0.1190	0.1210	0.1230	0.1251	0.1271	0.1292	0.1314	0.1335	0.1357
-1.00	0.1379	0.1401	0.1423	0.1446	0.1469	0.1492	0.1515	0.1539	0.1562	0.1587
-0.90	0.1611	0.1635	0.1660	0.1685	0.1711	0.1736	0.1762	0.1788	0.1814	0.1841
-0.80	0.1867	0.1894	0.1922	0.1949	0.1977	0.2005	0.2033	0.2061	0.2090	0.2119
-0.70	0.2148	0.2177	0.2206	0.2236	0.2266	0.2296	0.2327	0.2358	0.2389	0.2420
-0.60	0.2451	0.2483	0.2514	0.2546	0.2578	0.2611	0.2643	0.2676	0.2709	0.2743
-0.50	0.2776	0.2810	0.2843	0.2877	0.2912	0.2946	0.2981	0.3015	0.3050	0.3085
-0.40	0.3121	0.3156	0.3192	0.3228	0.3264	0.3300	0.3336	0.3372	0.3409	0.3446
-0.30	0.3483	0.3520	0.3557	0.3594	0.3632	0.3669	0.3707	0.3745	0.3783	0.3821
-0.20	0.3859	0.3897	0.3936	0.3974	0.4013	0.4052	0.4090	0.4129	0.4168	0.4207
-0.10	0.4247	0.4286	0.4325	0.4364	0.4404	0.4443	0.4483	0.4522	0.4562	0.4602
-0.00	0.4641	0.4681	0.4721	0.4761	0.4801	0.4840	0.4880	0.4920	0.4960	0.5000

Table G.2.: Cumulative standard normal distribution, z = 0.0 to 3.49

	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.00	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.10	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.20	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.30	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.40	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.50	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.60	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.70	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.80	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.90	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.00	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.10	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.20	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.30	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.40	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.50	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.60	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.70	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.80	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.90	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.00	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.10	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.20	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.30	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.40	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.50	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.60	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.70	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.80	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.90	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.00	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990
3.10	0.9990	0.9991	0.9991	0.9991	0.9992	0.9992	0.9992	0.9992	0.9993	0.9993
3.20	0.9993	0.9993	0.9994	0.9994	0.9994	0.9994	0.9994	0.9995	0.9995	0.9995
3.30	0.9995	0.9995	0.9995	0.9996	0.9996	0.9996	0.9996	0.9996	0.9996	0.9997
3.40	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9998

Table G.3.: Percentage Points $t_{\nu,\alpha}$ of the Student-t Distribution

$\nu \setminus \alpha$	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
∞	1.282	1.645	1.960	2.326	2.576	2.807	3.090	3.291

Table G.4.: Percentage Points $\chi_{\nu,\alpha}^2$ of the Chi-Square Distribution (ν) degrees of freedom

$\nu \setminus \alpha$	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

G. Statistical Tables

Table G.5.: Kolmogorov-Smirnov Test Critical Values

n	$D_{0.1}$	$D_{0.05}$	$D_{0.01}$
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>

Bibliography

- Ahrens, J. and Dieter, V. (1972). Computer methods for sampling from the exponential and normal distributions. *Communications of the Association for Computing Machinery*, 15:873–82.
- Albright, S. C. (2001). *VBA for Modelers, Developing Decision Support Systems with Microsoft Excel*. Duxbury Thomson Learning.
- Alexopoulos, C. and Seila, A. F. (1998). Output data analysis. In Banks, J., editor, *Handbook of Simulation*. John Wiley & Sons, New York.
- April, J., Glover, F., Kelly, J., and Laguna, M. (2001). Simulation optimization using real-world applications. In Peters, B. A., Smith, J. S., Medeiros, D. J., and Rohrer, M. W., editors, *The Proceedings of the 2001 Winter Simulation Conference*. Piscataway, New Jersey: Institute of Electrical and Electronic Engineers.
- Askin, R. G. and Goldberg, J. B. (2002). *Design and analysis of lean production systems*. John Wiley & Sons.
- Askin, R. G. and Standridge, C. R. (1993). *Modeling and Analysis of Manufacturing Systems*. John Wiley & Sons.
- Axsäter, S. (2006). *Inventory Control*. Springer Science + Business Media.
- Balci, O. (1997). Principles of simulation model validation, verification, and testing. *Transactions of the Society for Computer Simulation International*.
- Balci, O. (1998). Verification, validation, and testing. In *The Handbook of Simulation*, pages 335–393. John Wiley & Sons.
- Ballou, R. H. (2004). *Business logistics/supply chain management: planning, organizing, and controlling the supply chain*. Prentice-Hall, 5th edition.
- Banks, J. (1998). *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. John Wiley & Sons.
- Banks, J., Burnette, B., Kozloski, H., and Rose, J. (1995). *Introduction to SIMAN V and CINEMA V*. John Wiley & Sons.
- Banks, J., Carson, J., Nelson, B., and Nicol, D. (2005). *Discrete-Event System Simulation*. Prentice Hall, 4th edition.
- Biller, B. and Nelson, B. L. (2003). Modeling and generating multivariate time-series input processes using a vector autoregressive technique. *Assoc. Comput. Mach. Trans. Modeling and Comput. Simul.*, 13:211–237.
- Blanchard, B. S. and Fabrycky, W. J. (1990). *Systems Engineering and Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey.
- Bolch, G., Greiner, S., de Meer, H., and Trivedi, K. (2006). *Queueing Networks and Markov Chains*. John Wiley & Sons, 2nd edition.

Bibliography

- Box, G. E. P., Jenkins, G. M., and Reinsel, G. C. T. S. A. (1994). *Forecasting and Control*. Prentice Hall, 3rd edition.
- Box, G. E. P. and Muller, M. F. (1958). Note on the generation of random normal deviates. *Annals of Mathematical Statistics*, 29:610–11.
- Buzacott, J. A. and Shanthikumar, J. G. (1993). *Stochastic Models of Manufacturing Systems*. Prentice-Hall.
- Cario, M. C. and Nelson, B. L. (1996). Autoregressive to anything: Time series input processes for simulation. *Operations Research Letters*, 19:51–58.
- Cario, M. C. and Nelson, B. L. (1998). Numerical methods for fitting and simulating autoregressive-to-anything processes. *INFORMS Journal of Computing*, 10:72–81.
- Casella, G. and Berger, R. (1990). *Statistical Inference*. Wadsworth & Brooks/Cole.
- Cash, C. R., Dippold, D. G., Long, J. M., Nelson, B. L., and Pollard, W. P. (1992). Evaluation of tests for initial conditions bias. In Swain, J. J., Goldsman, D., Crain, R. C., and Wilson, J. R., editors, *Proceedings of the 1992 Winter Simulation Conference*, pages 577–585.
- Cheng, R. C. (1977). The generation of gamma variables with nonintegral shape parameters. *Applied Statistics*, 26(1):71–75.
- Chopra, S. and Meindl (2007). *Supply Chain Management: strategy, planning, and operations*. Prentice-Hall, 3rd edition.
- Chung, C. A. (2004). *Simulation Modeling Handbook: A Practical Approach*. CRC Press.
- Command, A. F. S. (1991). *ASD Directorate of Systems Engineering and DSMC Technical Management Department*.
- Cooper, R. B. (1990). *Introduction to Queueing Theory*. CEEPress Books, 3rd edition.
- Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag, New York.
- Fishman, G. S. (2001). *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer, New York.
- Fishman, G. S. (2006). *A First Course in Monte Carlo*. Thomson Brooks/Cole.
- Fishman, G. S. and Yarberry, L. S. (1997). An implementation of the batch means method. *INFORMS Journal on Computing*, 9.
- Glover, F., Kelly, J. P., and Laguna, M. (1999). New advances for wedding optimization and simulation. In Farrington, F., Nembhard, H. B., Sturrock, D. T., and Evans, G. W., editors, *The Proceedings of the 1999 Winter Simulation Conference*, Piscataway, New Jersey. Institute of Electrical and Electronic Engineers.
- Glynn, P. W. and Whitt, W. (1989). Extensions of the queueing relation and $L = \lambda W$ and $H = \lambda G$. *Operations Research*, 37:634–644.
- Goldsman, D. and Nelson, B. L. (1998). Comparing systems via simulation. In Banks, J., editor, *Handbook of Simulation*. John Wiley & Sons, New York.
- Gross, D. and Harris, C. M. (1998). *Fundamentals of Queueing Theory*. John Wiley & Sons, New York, 3rd edition.
- Gross, D., Shortle, J. F., Thompson, J. M., and Harris, C. M. (2008). *Fundamentals of Queueing Theory*. John Wiley & Sons.

Bibliography

- Hadley, G. and Whitin, T. M. (1963). *Analysis of Inventory Systems*. Prentice Hall.
- Henriksen, J. and Schriber, T. (1986). Simplified approaches to modeling accumulating and non-accumulating conveyor systems. In *Proceedings of the 1986 Winter Simulation Conference*. Institute of Electrical and Electronic Engineers.
- Hull, T. E. and Dobell, A. R. (1962). Random number generators. *SIAM Review*, 4:230–254.
- Kelly, F. P. (1979). *Reversibility and Stochastic Networks*. John Wiley & Sons.
- Kelton, W. D., Sadowski, R. P., and Sturrock, D. T. (2004). *Simulation with Arena*. McGraw-Hill, 3rd edition.
- Kelton, W. D., Sadowski, R. P., and Sturrock, D. T. (2015). *Simulation with Arena*. McGraw-Hill, 6th edition.
- Kendall, D. G. (1953). 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.
- Kingman, J. F. C. (1964). *The heavy traffic approximation in the theory of queues*. Proceedings of the Symposium On Congestion Theory.
- Kleijnen, J. P. C. (1988). Analyzing simulation experiments with common random numbers. *Management Science*, 34:65–74.
- Kleijnen, J. P. C. (1998). Experimental design for sensitivity analysis, optimization, and validation of simulation models. In Banks, J., editor, *Handbook of Simulation*. John Wiley & Sons, New York.
- Kleinrock, L. (1975). *Queueing Systems*, volume 1. John Wiley & Sons.
- Lada, E. K., Wilson, J. R., and Steiger, N. M. (2003). A Wavelet-Based Spectral Method for Steady-State Simulation Analysis. Proceedings of the 2003 Winter Simulation Conference.
- Law, A. (2007). *Simulation Modeling and Analysis*. McGraw-Hill, 4th edition.
- L'Ecuyer, P., Simard, R., and Kelton, W. D. (2002). An object-oriented random number package with many long streams and substreams. *Operations Research*, 50:1073–1075.
- Leemis, L. (1991). Nonparametric estimation of the cumulative intensity function for a nonhomogeneous poisson process. *Management Science*, 37(7):886–900.
- Leemis, L. M. and Park, S. K. (2006). *Discrete-Event Simulation: A First Course*. Prentice-Hall.
- Little, J. D. C. (1961). A proof for the queuing formula $L = \lambda W$. *Operations Research*, 9:383–387.
- Litton, J. R. and Harmonosky, C. H. (2002). *A Comparison of Selective Initialization Bias Elimination Methods*. Proceeding of the 2002 Winter Simulation Conference.
- Livny, M., Melamed, B., and Tsiolis, A. K. (1993). The impact of autocorrelation on queueing systems. *Management Science*, 39(3):322–339.
- Miller, L. H. (1956). Table of percentage points of kolmogorov statistics. *Journal of the American Statistical Association*, pages 111–121.
- Montgomery, D. C. and Runger, G. C. (2006). *Applied Statistics and Probability for Engineers*. John Wiley & Sons, 4th edition.

Bibliography

- Muckstadt, J. A. and Sapras, A. (2010). *Principles of Inventory Management: When You Are Down to Four, Order More*. Springer.
- Nahmias, S. (2001). *Production and Operations Analysis*. McGraw-Hill, 4th edition.
- Patuwo, B. E., Disney, R. L., and Mcnickle, D. C. (1993). The effect of correlated arrivals on queues. *IIE Transactions*, 25(3):105–110.
- Pegden, C. D., Shannon, R. E., and Sadowski, R. P. (1995). *Introduction to Simulation Using SIMAN*. McGraw-Hill, 2nd edition.
- Ravindran, A., Phillips, D., and Solberg, J. (1987). *Operations Research Principles and Practice*. John Wiley & Sons, 2nd edition.
- Ripley, B. D. (1987). *Stochastic Simulation*. John Wiley & Sons Inc.
- Robinson, S. (2005). Automated analysis of simulation output data. *Proceedings of the 2005 Winter Simulation Conference*, 763-770.
- Ross, S. (1997). *Introduction to Probability Models*. Academic Press, 6th edition.
- Rossetti, M. D. (2008). JSI: An open-source object-oriented framework for discrete-event simulation in Java. *International Journal of Simulation and Process Modeling*, 4(1):69–87.
- Rossetti, M. D. and Delaney, P. J. (1995). *Control of initialization bias in queueing simulations using queueing approximations*. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.
- Rossetti, M. D. and Li, Z. (2005). Exploring exponentially weighted moving average control charts to determine the warm-up period. In Kuhl, M. E., Steiger, N. M., Armstrong, F. B., and Joines, J. A., editors, *Proceedings of the 2005 Winter Simulation Conference*, pages 771–780, Piscataway, New Jersey. Institute of Electrical and Electronics Engineers.
- Rossetti, M. D. and Seldanari, F. (2001). Multi-objective analysis of Hospital Delivery Systems. *Computers and Industrial Engineering*, 41:309–333.
- Schmeiser, B. W. (1982). Batch size effects in the analysis of simulation output. *Operations Research*, 30:556–568.
- Schriber, T. J. and Brunner, D. T. (1998). How discrete-event simulation software works. In Banks, J., editor, *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, chapter 24. John Wiley & Sons, New York, New York.
- Silver, E. A., Pyke, D. F., and Peterson, R. (1998). *Inventory Management and Production Planning and Scheduling*. John Wiley & Sons, 3rd edition.
- Singh, N. (1996). *Systems Approach to Computer-Integrated Design and Manufacturing*. John Wiley & Sons.
- Soto, J. (1999). Statistical testing of random numbers. In *Proceedings of the 22nd National Information Systems Security Conference*.
- Stecke, K. E. (1992). Machine interference: Assignment of machines to operators. In Salvendy, G., editor, *Handbook of Industrial Engineering*, chapter 57. John-Wiley & Sons.
- Steiger, N. M. and Wilson, J. R. (2002). An improved batch means procedure for simulation output analysis. *Management Science*, 48.
- Tijms, H. C. (2003). *A First Course in Stochastic Models*. John-Wiley & Sons.

Bibliography

- Welch, P. D. (1983). 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.
- White, K. P., Cobb, M. J., and Spratt, S. C. (2000). *A Comparison of Five Steady-State Truncation Heuristics for Simulation*. Proceeding of the 2000 Winter Simulation Conference.
- Whitt, W. (1983). The queueing network analyzer. *The Bell System Technical Journal*, 62(9):2779–2815.
- Whitt, W. (1989). Planning queueing simulations. *Management Science*, 35(11):1341–1366.
- Whitt, W. (1993). Approximations for the GI/G/m queue. *Productions and Operations Management*, 2(2):114–161.
- Wilson, J. R. and Pritsker, A. A. B. (1978). A survey of research on the simulation startup problem. *Simulation*.
- Zipkin, P. H. (2000). *Foundations of Inventory Management*. McGraw-Hill.

Index

across replication statistics, 135
activity diagram, 15

conceptual model, 16

DEGREE, 11

experimental design, 12

iteration, 11

model

- descriptive, 8
- predictive, 8
- prescriptive, 8

model translation, 16

problem formulation, 12

randomness, 9

simulation, 1

- advantages, 4
- definition, 2
- languages, 9
- methodology, 11

system, 4

- conceptualization, 6
- continuous, 6, 7
- definition, 4, 14
- deterministic, 6
- discrete, 6, 7
- dynamic, 6
- static, 6
- stochastic, 6
- types, 6

trade-off, 17

validation, 12, 16

verification, 12, 16

within replication statistics, 129

world view, 5